University of Birmingham

Department of Electronic, Electrical and Computer Engineering

UNIVERSITY OF BIRMINGHAM

MSc Dissertation

An autonomous, co-operative tele-robotic agent framework for WowWee Rovio

Konstantinos Tsimpoukas

I.D. 1130191

MSc Embedded Systems

Supervisor: Theo Arvanitis

September 2011

Birmingham, UK

ACKNOWLEDGEMENTS

I would like to thank Dr Theodoros N. Arvanitis, my dissertation supervisor, for letting me choosing an interesting and challenging project. Also, I would like to thank the Department of Electronic, Electrical and Computer Engineering of the University of Birmingham, and the Centre for Learning, Innovation and Collaboration for providing the appropriate tools and equipment to fulfil successfully my MSc final project.

Many thanks to my family and friends that they were there when I needed them and for the support that they always provide me.

ABSTRACT

This project presents a telerobotic agent framework for WowWee Rovio. An advanced and parametrical Graphical User Interface has been implemented. Through this GUI the operator can manually command the available agents or can trigger agents to complete tasks by applying them autonomous behaviours. For the completion of these tasks, that require object recognition techniques, advanced image processing methods were developed. The design and the implementation of advanced tasks were possible after developing and completing successfully smaller and simpler tasks. The GUI allows the intervention of the user applying semi-autonomous behaviour for specific functionalities in order to help agents to successfully fulfil autonomous co-operating tasks. Experimental results and measurements of this framework demonstrate that the Rovios can successfully and efficiently complete the various tasks within a reasonable amount of time meeting all the constraints that have been set.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 TOPICS

1.1.1 TELE-PRESENCE ROBOTS

It is stated that an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors (Norvig, et al., 1995). A human agent has eyes, ears and other organs for sensors while a robotic agent has cameras and infrared sensors. The effectors for a human are the various body parts and for the robotic agent the motors.

The term “telepresence robot” encompasses a wide range of robots, which vary with respect to mobility, size and capabilities. A set of guidelines has been suggested that is believed that are essential for all telepresence agents (Holly, et al., 2011). These guidelines are not merely a list of desirable features, but, rather, they constitute an essential set of features that must be incorporated in telepresence robots.

- Video information is critical in telepresence robots for conversation and navigation. Due to the mobility afforded by these robots, the information must be transferred wirelessly. Video streams constitute a significant portion of the transferred data and can be adversely affected by the network connection.
- Audio quality is most important component of communicating through a telepresence robot is the conversation itself. The audio quality must be comparable to that of a landline phone conversation.
- The user interface is a critical component of the telepresence system. It is the driver’s portal to the remote world. The UI must be simple, easy to use, not distracting, and provide the necessary functionality without overwhelming the driver. User interfaces for controlling remote robots have been well researched (Drury, et al., 2010).
- Physical features:
    - Robot height: Ideally, the driver should be able to change the robot’s height to any desired length remotely.
    - Robot speed: The robots should be able to move at average human walking speeds of about 3 miles per hour.
    - Wide field of view: The front facing camera must have a wide field of view (FOV). A wide FOV is essential during navigation because it provides the driver with better situation awareness.
    - A head that could pan and/or tilt.

- Access point switching: As robots move around, they have to switch access points. Depending on the environment, there can be multiple access points that the robot might have to connect to while moving.

- Autonomous navigation behaviours are desirable because of safety reasons and for ease of use. For instance, a remote driver may see in their video feed a person exiting a conference room and stop so the person can pass in front of the robot. Under teleoperation, the robot may or may not stop in time, depending on the delay given the robot's video feed to the driver and then the navigation command back to the robot. Processing the sensor data locally allows the robot to take immediate action, thereby providing a tighter closed loop control of the robot. Hence, autonomous behaviours allow for better control of the robot under varying network conditions.
- Social considerations. The previous guidelines provide the technical and functional competence of a telepresence robot. Social acceptance will also be required for long-term acceptance.

The WowWee Rovios that have been used for the implementations of this project are commercially available telepresence robotic agents.

1.1.2 Agent Interaction and Co-operation

Interaction is another concept that concerns us. Typically, interaction is viewed as any influence that affects an agent's behaviour. By this definition, an agent interacts with everything it can sense or be affected by, since all of its external and internal state can have an impact on its actions (Mataric, 1994).

Cooperation is a form of interaction, usually based on communication. Certain types of cooperative behaviour depend on communication. Specifically, any cooperative behaviour that requires negotiation between agents depends on directed communication in order to assign particular tasks to the participants.

1.1.3 Robot Vision

This project is greatly concerned with robot vision. Although vision is effortless for humans, it has been proven to be a very difficult problem for machines. Major sources of difficulty include variable and uncontrolled illumination, shadows, complex and hard-to-describe objects such as those that occur in the natural environments. These difficulties are reduced in man-made environments such as the interior of a building or an office. Computer vision in these places has been proven to be much more successful (Nilsson, 1998).

It is stated that the goal of computer vision research is to provide computers with human like perception capabilities so that they can sense the environment, understand the sensed data, take appropriate actions, and learn from this experience in order to enhance future performance (Ashutosh, et al., 2005).

Navigation, target recognition, manufacturing, photo interpretation or remote sensing are real world applications which require vision algorithms and systems to work under partial occlusion, possibly under high clutter, low contrast, and changing environmental conditions. This requires that the vision techniques should be robust and flexible to optimize performance in a given scenario (Ashutosh, et al., 2005).

1.1.4 Machine Learning

Machine Learning is an area of Artificial Intelligence which aims to develop techniques allowing machines to "learn". Thus Machine Learning is interested in finding possible methods to make computers using experience to act more rationally than if they didn't have any knowledge of the past. Acting rationally is acting in order to achieve the best outcome or the best expected outcome, if there is uncertainty. Finally, the field of machine learning is driven by the idea that computer algorithms and systems can improve their own performance with time.

1.2 Inspiration

The previous chapters explained the main topics that this project is concerned about. The following two examples are works and implementations worldwide known which inspired me to study and investigate fields like Artificial Intelligence and Robotics.

1.2.1 Robot World Cup

The Robot World Cup Initiative is an attempt to foster AI and intelligent robotics research by providing a standard problem where wide range of technologies can be integrated and examined. The first RoboCup was held in Nagoya at 1997. In order for a robot team to perform a soccer game technologies like autonomous agents, multi-agent collaboration, strategy acquisition, real time reasoning, robotics and sensor fusion have to be incorporated (Asada, et al., 1997). Exceptionally interesting is the main goal of the RoboCup, which is stated in the original RoboCup website,:

"By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup."

Figure 1-1 RoboCup

Figure 1-1 shows some robots in action during RoboCup. As you can see there are different kinds and sizes of robots which result in different categories of competing. The main categories are Humanoid, Middle Size, Simulation, Small Size and Standard Platform.

1.2.2 Deep Blue

Deep Blue is the chess playing machine which defeated World Chess Champion Garry Kasparov during their 1997 Rematch. Deep Blue started as Chiptest at Carnegie Mellon University and by the time that won Kasparov was the top IBM research project. In 1989 IBM recruited three Carnegie Mellon University Ph.D. graduates to create a chess playing computer that would outlay the best human on planet Earth.

It is identified as a monumental triumph, ranking as one of the great technological achievements of the 20th century (Newborn, 2004) and pushed Artificial Intelligence in higher levels.

The challenge of this attempt is obvious by recalling that in the typical chess position there are approximately 30 moves. To each of these 30 moves, there are 30 approximately replies, leading to 900 different positions after one move by each side. Rounding the number to 1000, to make the calculations simpler, we find out that looking two movements ahead by each side leads to about 1000 times 1000, or about 1,000,000 positions. To play chess at the level of a world champion the computer has to be able to look 12 positions ahead which correspond to an incredible 1,000,000,000,000,000,000 positions. Moreover, the computer has to search many of these positions deeper in order to follow tactical variations (Newborn, 2004).

1.3 Motivation

The previous chapters explained the basic principles of the domains that this project is about. Trying to understand how multi-agent systems co-operate and interact is a very interesting domain which is gradually gaining popularity assisting people in their daily and labour life.

On a practical point of view, building a framework of co-operating agents, even with simple and basic behaviours, uncovers practical problems during the implementations. Data transmission, quality of connection and programming methods are some of them.

From a theoretical point of view, studying the behaviour of co-operating agents and their responses in a dynamic environment leads in developing more robust multi-agent co-operating frameworks. From the other side, the exploration of methods in computer vision helps us understand the current problems and also what lies ahead. Machine learning and human-robot interaction creates semi-autonomous frameworks which will gradually become completely autonomous.

1.4 Scope of the Project

The main scope of this project is to build an autonomous, co-operative tele-robotic agent framework for WowWee Rovio's.

1.4.1 Objectives

Analysing the scope deeper down, six basic objectives can be distinguished:

1. Study the WowWee Rovio's autonomous behaviour.
2. Build an interface for the WowWee Rovio to communicate to an external party and/or other Rovio's.
3. Achieve a basic function of autonomous exploration in unfamiliar environments.
4. Use of advanced image processing algorithms to identify and describe objects within unfamiliar environments.
5. Coding a knowledge base to store the robot's experience.
6. Achieve basic communication and co-operation between at least two robotic agents.

1.5 Structure of Dissertation

Chapter 1 presented the main areas that this project is concerned. Also, two projects were presented that inspired and motivated me to complete this work. Finally, the main aim and objectives of this dissertation were given.

Chapter 2 presents related works that lead the described areas of the introductory chapter.

Chapter 3 describes the main characteristics and features of the Rovios and focuses on those that were useful to this project.

Chapter 4 presents and describes the methods and the techniques that were explored and implemented in this framework for the Rovios.

Chapter 5 shows the implementations and the way that they were developed, describing at the same time their characteristics and abilities.

Chapter 6 tests and evaluates the various methods that were developed in this project.

Chapter 7 concludes discussing the results and the effectiveness of the previously described methods.

2 LITERATURE REVIEW

2.1 RELATED WORK

In the first chapter the main areas that are related to this project were presented; Tele-presence , Agent Interaction and Co-operation, Robot Vision and Machine Learning. Related work that leads these areas of interest is presented below.

2.1.1 THE ROBOCUP

The RoboCup competition (Asada, et al., 1997) combines all the above described areas. The robots are wirelessly interacting and co-operating to succeed their goal. Furthermore, a good strategy has to be implemented and followed by them. In this case they are trying to score against the other team of robots. Robot vision and motor control are also very critical in their performance as they have to recognise the ball (usually an orange ball), the borders of the field and the area of the goalpost then position themselves accordingly around the ball and finally "kick" the ball successfully in the right direction. Another important parameter is the environment where they have to act. The environment continuously changes as the robots and the ball change positions on the field.

There are several different competing categories. The standard platform league teams use identical (standard) robots. Therefore, the teams concentrate on software development only, while still use state-of-the-art robots, as it happen in this project. We are not concerned on the hardware but just on the method development. This gives us the opportunity to focus on the development and testing methods using more than one agent.

This league replaced the highly successful four-legged league, based on Sony's AIBO dog robots, and is now based on Aldebaran's Nao humanoids.

The RoboCup competition has two leagues, the "real" and the "virtual" simulation league. In RoboCup's "virtual" competition, players are not robots but computer programs which manipulate virtual robots through RoboCup's provided simulator, the RoboCup Soccer Server (Itsuki, 1995). The Soccer Server provides a simulator environment with complex dynamics, noisy and limited sensor information, noisy control, and real-time play (Figure 2-1). To win a soccer match in the Soccer Server, players must overcome these issues and cooperate as a team in the face of limited communication ability and an incomplete world-view (Farris, et al., 1998).

It has been suggested that genetic programming is a promising new method for automatically generating functions and algorithms through natural selection (Farris, et al., 1998). In contrast to

other learning methods, genetic programming's automatic programming makes it a natural approach for developing algorithmic robot behaviours.

Figure 2-1 Virtual RoboCup

Farris et al., (1998) studies the evolution of the behaviours of the virtual agents and says that one of the benefits of working with evolutionary computation is being able to watch the population learn. It is also stated that their initial random teams consisted primarily of players which wandered aimlessly, spun in place, stared at the ball, or chased after teammates. Early populations produced all sorts of bizarre strategies. The "kiddie-soccer" was a problematic strategy where everyone on the team would go after the ball and try to kick it into the goal. After a number of generations, the population as a whole began to develop rudimentary defensive ability. Eventually teams began to disperse players throughout the field and to pass to teammates when appropriate instead of kicking straight to the goal.

Salustowicz et al., (1998) simulated soccer to study multiagent learning. Each agent shares action set and policy, but may behave differently due to position-dependent inputs. The agents of the same team are punished or rewarded in case of goals. They conducted simulations with varying team sizes and compared several learning algorithms. The main algorithms that were tested were the TD-Q learning with linear neural networks, Probabilistic Incremental Program Evolution (PIPE) and a PIPE version which learns by coevolution (CO-PIPE). TD-Q is based on learning evaluation functions (EFs) mapping input/action pairs to expected reward. PIPE and CO-PIPE search policy space directly. The result of the simulations showed that PIPE and CO-PIPE learn faster than linear TD-Q and continuously increased their performance. This suggests that PIPE-like, EF-independent techniques can easily be applied to complex multiagent learning scenarios with policy shared agents, while more sophisticated and time consuming EF-based approaches may be necessary to overcome TD-Q's current problem (Salustowicz, et al., 1998).

Stone and Veloso (1998) introduce the level layered behaviour. They describe two levels of learned behaviours. First, the clients learn a low-level individual skill that allows them to control the ball effectively. Then, using this learned skill, they learn a higher level skill that involves multiple players. The kicking of the ball by an agent is a low level skill which is a prerequisite to more complicated behaviours. This skill is a form of simple Multiagent Learning. Such an action makes sense only in an environment where more agents co-exist and co-operate. Extending this behaviour to passing the ball to another teammate, they finally implemented a set of play involving several players and several uses of the learned behaviours (Stone, et al., 1998).

2.1.2 The Behaviour-Based Approach

Behaviour is a reaction to a stimulus. Arkin (1998) suggests that this pragmatic view enables us to express how a robot should interact with its environment. The reactive robotic systems have the following characteristics (according to (Arkin, 1998)):

- Behaviours serve as the basic building blocks for robotic actions.
- Use of explicit abstract representation knowledge is avoided in the generation of a response.
- Animal models of behaviour often serve as a basis for these systems.
- These systems are inherently modular from a software design perspective.

2.1.3 Ball Detection Techniques

Ball recognition is one of the most important and crucial tasks that the participant robots have to fulfil in RoboCup. Except from RoboCup, implementing a ball recognition method helps us to investigate the behaviour of multiple co-operating robots.

Many researchers are developing new techniques for ball detection for various environmental and lighting. Eventually the ball detection algorithms can either focus on a specific color search which is contrasting the general background and/or on circular shape detection.

A method for detecting and tracking the ball in a RoboCup scenario without the need for color information is being suggested in (Masselli, et al., 2004). They use Haar-like features trained by an adaboost algorithm to get a colorless representation of the ball. Tracking is performed by a particle filter and It is shown that the algorithm is able to track the ball in real-time with 25 fps even in a cluttered environment.

In their paper (Frintrop, et al., 2005), a new combination of a biologically inspired attention system (VOCUS - Visual Object detection with a CompUtational attention System) with a robust object detection method is presented. As an application, they built a reliable system for ball

recognition in the RoboCup context. Firstly, VOCUS finds regions of interest generating a hypothesis for possible locations of the ball. Secondly, a fast classifier verifies the hypothesis by detecting balls at regions of interest. The combination of both approaches makes the system highly robust and eliminates false detections. Furthermore, the system is quickly adaptable to balls in different scenarios: the complex classifier is universally applicable to balls in every context and the attention system improves the performance by learning scenario-specific features quickly from only a few training examples.

In (Bach , et al., 2005), they selected a simple pattern matching strategy to find ball candidates. The color of the ball in RoboCup is known, so, a single orange pixel was said to be a candidate for a ball. The ball identificator works in two steps: The first step tries to find a point near the centre of the visible part of the ball. The second step starts from the point obtained by step one to scan into eight directions for a transition from orange to green, white, sky-blue or yellow. If such a transition is found, the point where the transition takes place is marked as a candidate to lie on the outline of the ball. A transition from orange to another color stops the scanning process in that direction, and the point is not marked, because it is likely to signify a partly occluded edge of the ball. Finally, they succeed a rate of 25 fps without down-scaling the original image.

Martins et al. (2006) proposes a solution to detect standard FIFA balls, independent of their color, in the context of the RoboCup Middle Size League. The proposed approach is based on the use of an edge detection algorithm followed by the use of the circular Hough transform. Three edge detection operators were used in this experiment, Canny, Sobel and Laplace. Their experimental results show that the Canny edge detector is the best choice among the other edge detection algorithms, considering the blur effect resulting from the movement of the ball. Additionally, the Hough transform revealed to be a good method to detect circular shaped objects, and showed to be very tolerant to gaps in feature boundary descriptions and is relatively unaffected by image noise.

Another ball detection algorithm based on color information and Hough transform is being presented in (Zhang, et al., 2009). The Retinex algorithm is first applied to enhance the image. Then, an adaptive Hough transform is introduced to locate the position of the balls in order to reduce the influence of the complex environments. Thirdly, to detect the interested balls under different illumination conditions, a color invariant model Grayworld Normalization is introduced to overcome the influence of the illumination. The accuracy of this method is found to be 80.3%. The detection of some interested balls fails because the illumination conditions are so extreme that makes the ball to lose the color information.

2.1.4 User Interfaces for Robots

The interface between the user and the robot is a crucial factor in the success of their mission. Human and robot must co-operate, especially when the human operator and the robot are in remote places, in an easy and effective way. The operator must be aware of the robot's condition, state and features in order to be helpful in case of any possible collision.

Drury et al. (2010) suggest that remote robot interfaces can be partitioned into two categories: map-centric and video-centric. A map centric interface is an interface in which the map is the most predominant feature in the interface and most of the frequently used information is clustered on or near the map. Similarly, in a video-centric interface, the video window is the most predominant feature with the most important information located on or around the video screen.

MITRE Corporation developed a map-centric interface (Figure 2-2) which can involve up to three robots to map the area that the robots cover. The upper part of the interface was a map which was updated from the information that the robots were providing. The interface also had the ability to switch operator driving controls among the three robots. Also, small video windows from the robots appeared under the map. The main disadvantages of this interface are the slow rate of updates and the small size of the video screen (Drury, et al., 2003).

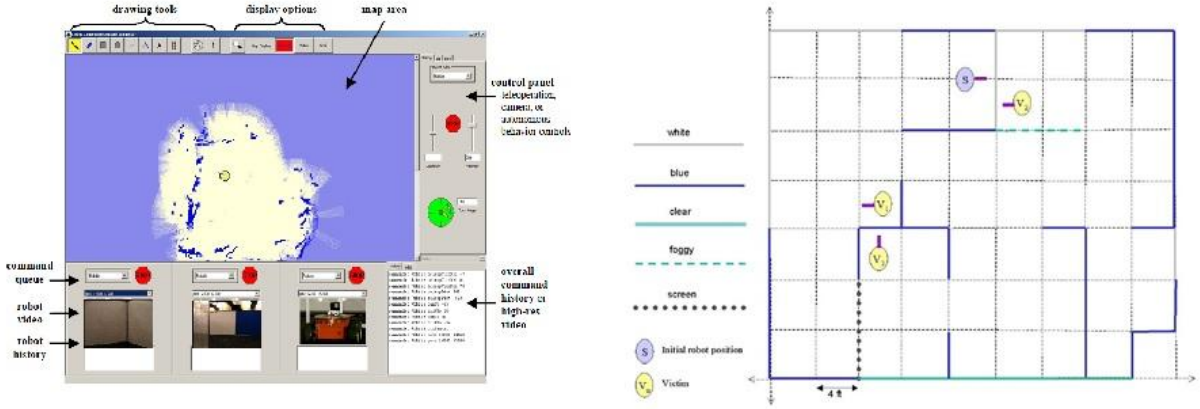

Figure 2-2 Command Console and Mapping interface (Drury, et al., 2003)

ARGOS from Brno University of Technology is an excellent example of a video-centric interface (Zalud, 2006).

The CASTER interface (Figure 2-3) developed at the University of New South Wales (Kadous, et al., 2006) also provides a full screen video interface but incorporates a different arrangement of small sensor feeds and status readouts placed around the edges. This human-robot interface was deployed by Team CASualty in the 2005 RoboCup Rescue Robot League competition. Some of the unique characteristics of CASTER interface are overlays of different sensors such as thermal cameras, integration of victim and landmark placement in 3D, use of 3D direction indicators for a

more consistent user experience, the use of hotkeys for camera placement, the display of accelerometer data, the addition of the auxiliary cameras, and doing away with numbers and text on the display altogether.

Figure 2-3 CASTER interface (Kadous, et al., 2006)

The GUI in Figure 2-4 was built and used by Gopalakrishnan et al., (2005) for their research. From this interface the user can access sonar and laser range data and live camera images and can manually control the robot and camera head. When an object of interest appears on live images, the user clicks on it in the video screen. Then the object is centered and zoomed for better view. After that, the automatic object recognition system is activated to recognize the object. If the robot cannot recognize the object, it directly asks the user for help. Speech synthesize (text-to-speech) is used for this purpose. The robot then uses its speech recognition capability to recognize what the user says.

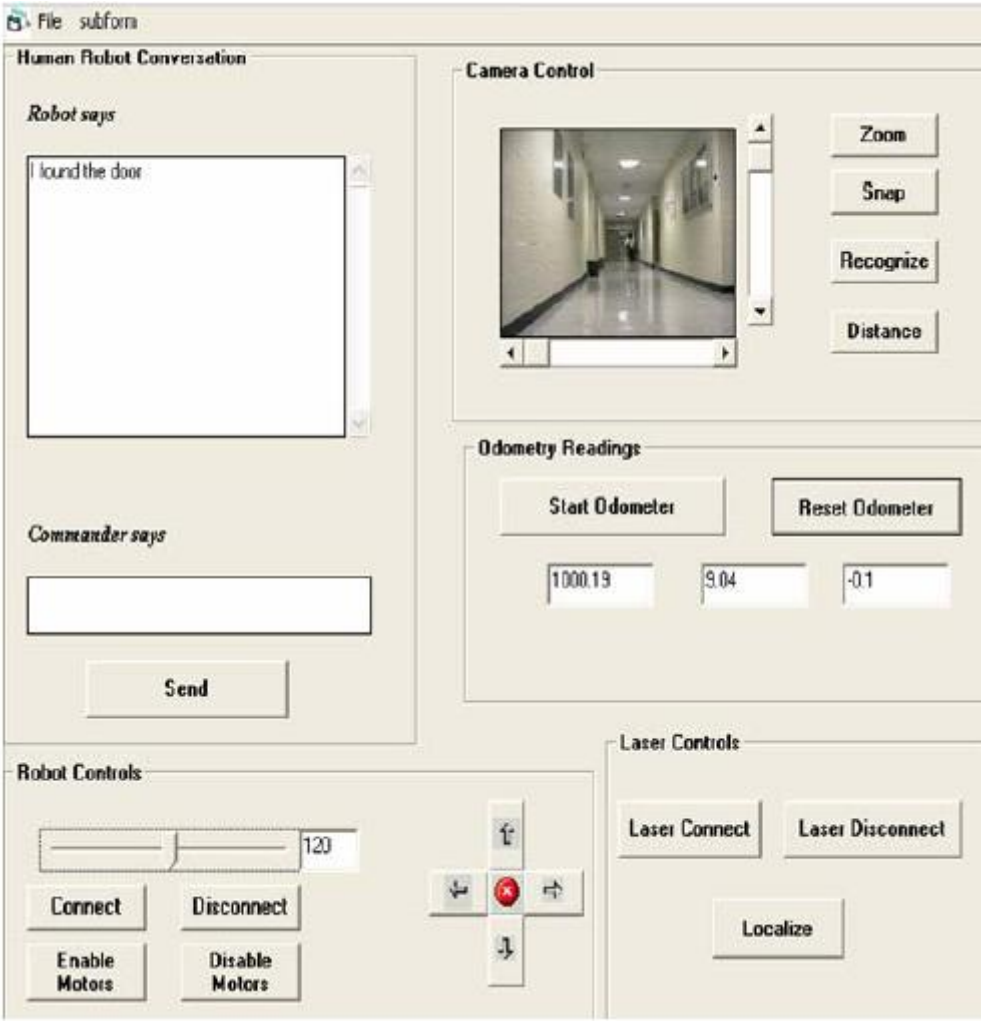

Figure 2-4 The graphical user interface for HRI (Gopalakrishnan, et al., 2005)

2.1.5 NAVIGATION AND SERVOING

One of the principal uses of vision is to provide information for manipulating objects as well as navigating in a scene while avoiding obstacles. Mobile robots moving around in an environment need to know where the obstacles are and where the free space corridors are available.

Russell and Norvig (1995) identify five major classes of algorithms of navigation and motion planning, and arrange them roughly in order of amount of information required at planning time and execution time:

- **Cell decomposition** methods break continuous space into a finite number of cells, yielding a discrete search problem.
- **Skeletonization** methods compute a one dimensional "skeleton" of the configuration space, yielding an equivalent graph search problem.
- **Bounded-error planning** methods assume bounds on sensors and actuator uncertainty and in some cases can compute plans that are guaranteed to succeed even in the face of severe actuator error.

- **Landmark-based navigation** methods assume that there are some regions in which the robot's location can be pinpointed using landmarks, whereas outside those regions it may have only orientation information.
- **Online algorithms** assume that the environment is completely unknown initially, although most assume some form of accurate position sensor.

A simple visual servoing and navigation algorithm for guiding a holonomic or omnidirectional robot based on landmarks is presented in (Begum, et al., 2010). The algorithm facilitates a mobile robot equipped mainly with a webcam to autonomously navigate an unknown environment and explore the path from start configuration to goal configuration along some checkpoints (landmarks) while avoiding obstacles. For their experiments they used the same robotic platform, the WowWee Rovio.

The logic of their vision algorithm is described below (also in Figure 2-5):

1. Detect red or green cones in image taken by the robot.
2. Convert the image into binary image where the detected cones are represented by regions of white pixels.
3. Bound the white regions by rectangles.
4. Calculate the areas of those rectangles and select the closest cone by simply taking the rectangle having largest area.
5. Find the centre of mass of selected region.
6. Find the state derived from the information of area, centre of the largest rectangle and presence of white pixel regions in the binary image.
7. Return the state that represents the position of the red or green cone in the environment with respect to robot viewpoint.

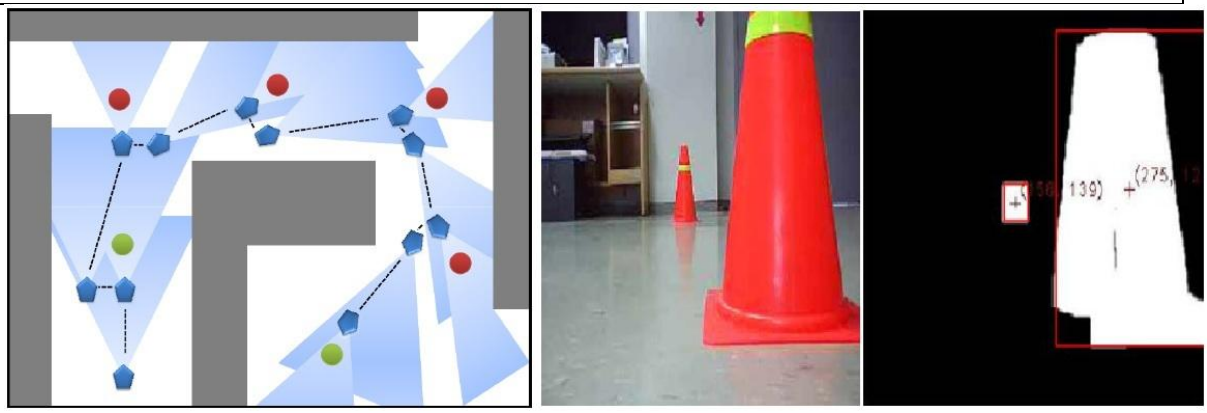

Figure 2-5 Navigation scenario and execution of the vision algorithm displaying the mass centre of the detected cone in the binary image (Begum, et al., 2010).

Figure 2-5 sketches the implemented scenario. Two green cones indicate the starting point while the red cones represent intermediate checkpoints/landmarks. The robot starts when it recognises a green cone. After that searches for red cones and moves towards the closest. Following the red cones it is finally stops when it reaches the ending green cone.

Gopalakrishnan et al. (2005) develop a vision-based learning mechanism for semi-autonomous mobile robot navigation. Laser-based localization, vision-based object detection and recognition, and route-based navigation techniques for a mobile robot have been integrated. Initially, the robot can localize itself in an indoor environment with its laser range finder. Additionally, a user can teleoperate the robot and point the objects of interest via a graphical user interface. Here the robot is semi-learning (automatically or with the help of the user) about landmarks and can navigate in a partially known environment (Figure 2-6).

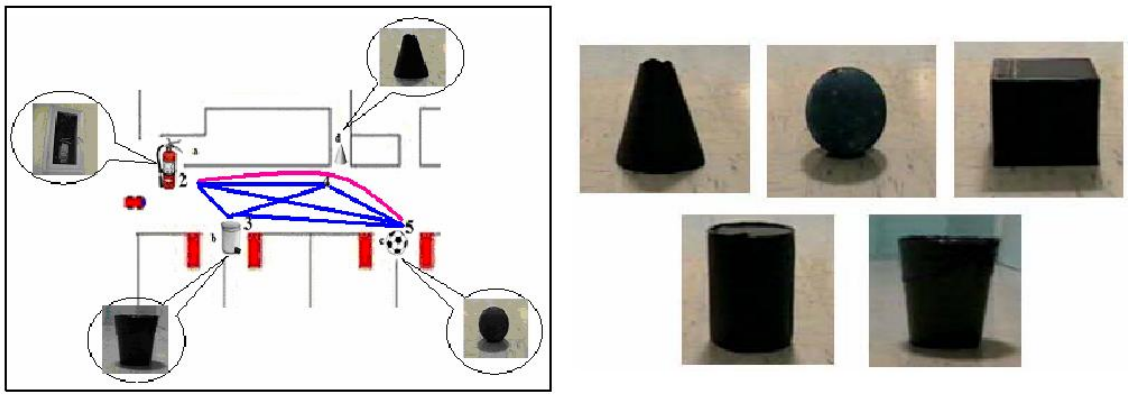

Figure 2-6 Mapping the routes and the object-landmarks that the robot recognised (Gopalakrishnan, et al., 2005).

2.2 SUMMARY

This chapter presented a summary of the relevant work in the different areas of research linked to the problem that has been chosen, and showed the importance of digital image processing and other sensors in creating a framework for autonomous co-operative agents in dynamic and unfamiliar environments.

The following chapter will present Rovio's main characteristics and which of them played a significant role in this project's methods and implementations.

3 WOWWEE ROVIO

Rovio is the Wi-Fi equipped mobile webcam that enables you to view and interact with its environment through live streaming video and audio with its built-in camera.

3.1 MAIN FEATURES

After setting Rovio and giving it a specific IP address you can access the web user interface by typing this IP on any browser. Through this interface you can manually control Rovio while you are getting the MJPEG steam from his built-in camera. Furthermore, you can record new paths and then it can play back these paths using the TrueTrack™ Navigation System. Rovio is checking the level of the battery and when it detects that the level is too low it can navigate back to its charging base.

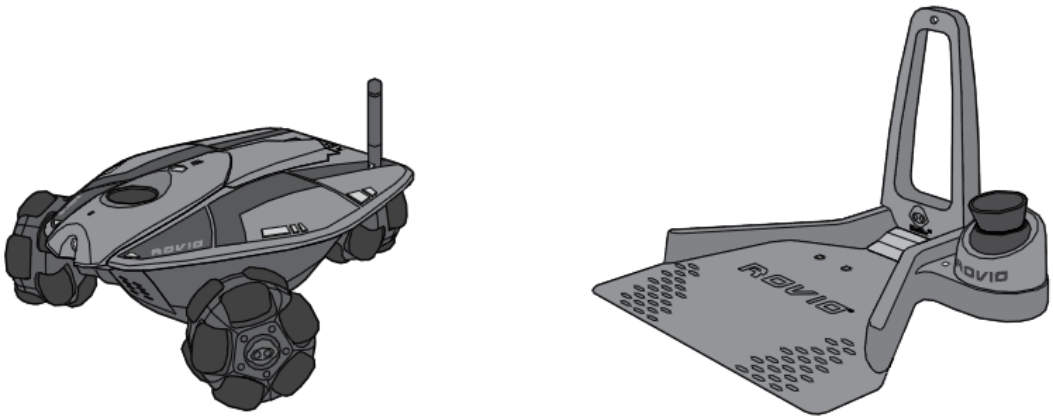

Figure 3-1 Rovio™ and Charging dock with built-in TrueTrack™ Beacon (taken from WowWee Group Limited, Rovio user's manual, 2009)

Figure 3-2 shows in greater detail the features of the WowWee's agent.

The USB connector is used basically to set up Rovio for the first time and adds it to the local wireless network. More than one Rovios can be connected. A different access IP is given each time that one Rovio is being set up. This project uses five Rovios. The number of the Rovios and their correspondent IPs are shown in Table 3-1.

Table 3-1 Rovios and corresponding IPs

Name	IP address
Rovio 1	http://192.168.2.11
Rovio 2	http://192.168.2.12
Rovio 3	http://192.168.2.14
Rovio 4	http://192.168.2.15
Rovio 5	http://192.168.2.16

The power button has a LED which indicates the current network connection and battery status of the current agent.

The neck of the agent can be in three positions (normal/down, mid and high). This characteristic makes Rovio more playful and at the same time allows the developer to use his creativity and implement in Rovio algorithms and methods like face detection or gesture recognition.

An antenna is an appropriate component for a wireless device like Rovio.

Microphone can be extremely useful especially in Speech Recognition applications.

The six blue LED indicators can be adjusted and also their brightness indicates if Rovio is charging or not when it is docked.

The IR sensor is used mainly for object avoidance. It is one of the main characteristics of the agent and its use will be analysed later on.

The built-in camera and stream of Jpeg images play a crucial role in the developed methods, mainly because these methods are based on image processing algorithms.

Finally, the omni-directional wheels allow movement in all directions and add characteristics in Rovio's kinetics like rotation and spinning.

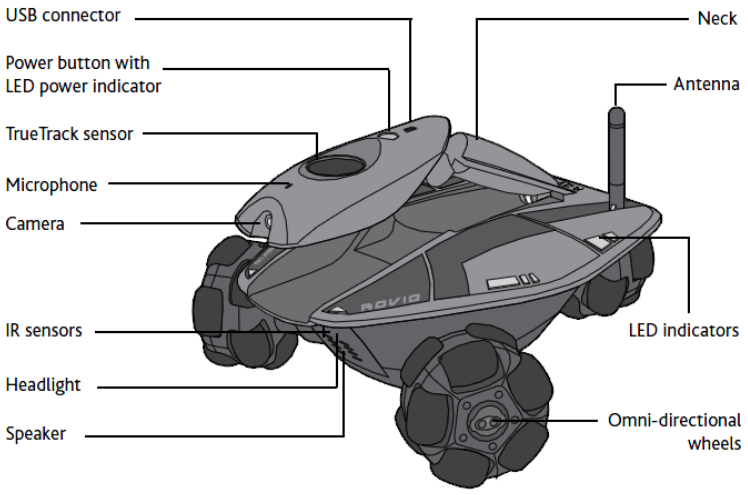

Figure 3-2 Rovio's main features and Sensors (taken from WowWee Group Limited, Rovio user's manual, 2009)

3.2 FEATURES USEFUL IN THIS PROJECT

Characteristics of this off-the-shelf toy robot are being presented in this sub-chapter that were useful in this project.

3.2.1 WIRELESS COMMUNICATION

The way that the robot communicates with an external application can be understood by studying Rovio's API and the existing Rovio web interface. Rovio is taking its command through the Uniform Resource Locator (URL) string using the Hypertext Transfer Protocol (HTTP). The wireless communication makes use of the 802.11b/g protocols (Figure 3-3 and Figure 3-4).

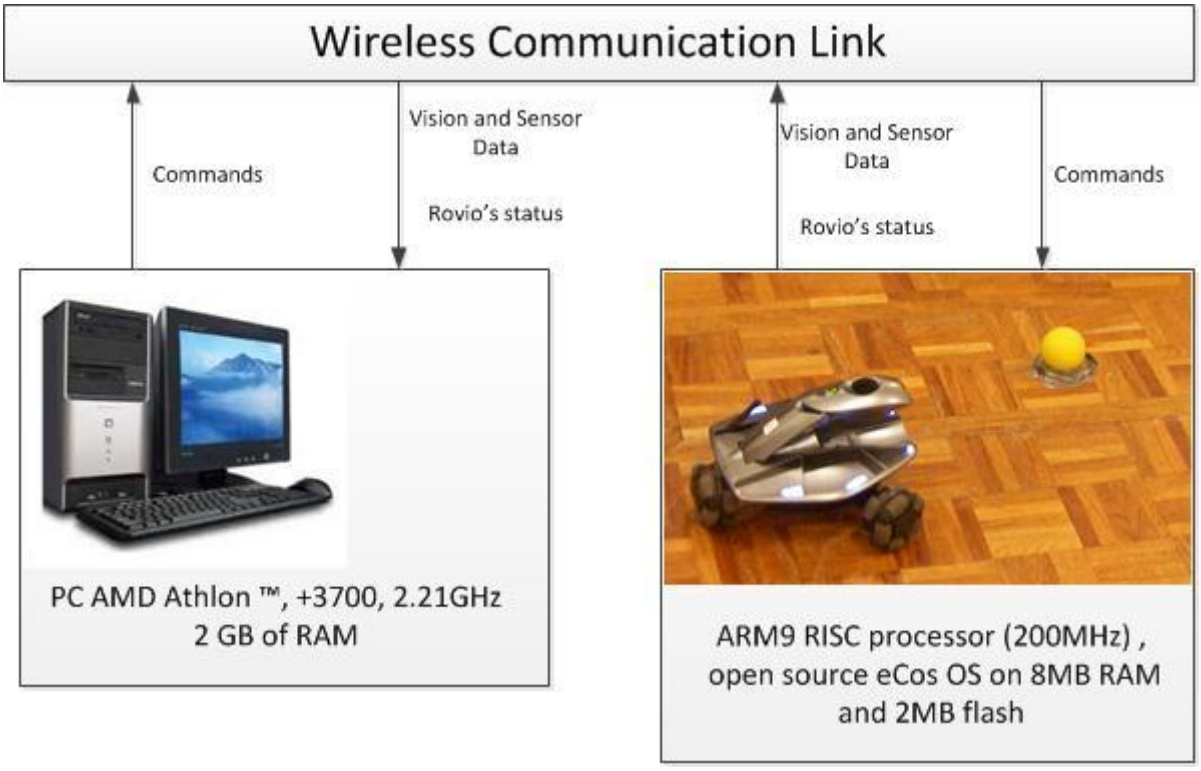

Figure 3-3 Experimental setup

3.2.2 FLEXIBLE PROGRAMMING

Rovio can be controlled by sending HTTP commands to a web server hosted on the robot (Figure 3-4). This allows controlling the robot using the sensors that the robot has and building AI algorithms without worrying about hardware configurations, setup and interfacing in high level programming language. All the methods and algorithms have been programmed in Visual C# in Visual Studio 2008 which operates in Windows XP on a +3700 AMD Athlon 2.21GHz PC. The proposed techniques implemented web based Rovio API v.1.3. Finally, the Open Source Computer Vision libraries (v.2.1) were used for the image processing part of the project.

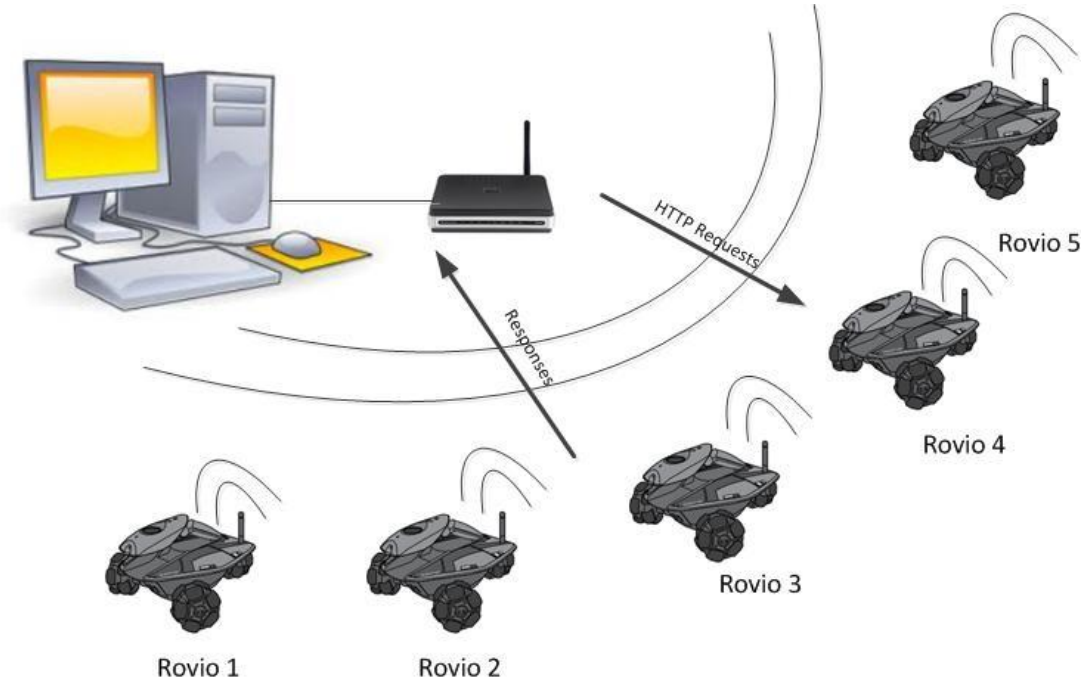

Figure 3-4 Way of Communication

3.2.3 Omni-directional Wheels

The omni-directional mobile robot systems have been popularly employed in several applications, particularly in the area of military services and surveillance and space exploration because of its capability to drive in all directions. Intensive research works on holonomic mobile robot systems have drawn much attention over two decades. Omnidirectional vehicles have great advantages over conventional (non-holonomic) platforms, with car-like Ackerman steering or differential drive system, for moving in tight areas. They can crab sideways, turn on the spot, and follow complex trajectories. These robots are capable of easily performing tasks in environments with static and dynamic obstacles and narrow aisles (Kim, et al., 2000) (Begum, et al., 2010).

Thanks to its special wheels Rovio is able to move on any direction, forward/backward but also sideways left/right, and turning on the spot (Figure 3-5). This is especially helpful when having to manoeuvre in a tight environment such as the man-made room.

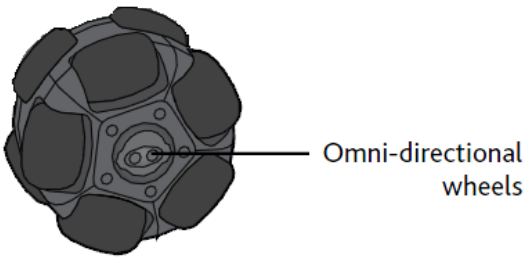

Figure 3-5 Omni-directional wheels of Rovio (taken from WowWee Group Limited, Rovio user's manual, 2009)

The holonomic feature of Rovio greatly simplifies the navigation problem.

3.3 SUMMARY

This chapter discussed the main features of Rovio and explained why this off-the-shelf toy can be useful in projects like this one that explores the areas of Machine Learning, Machine Vision and Agent Interaction and Co-operation.

The next chapter will describe the logic and the techniques that were used in this project.

4 TELEPRESENCE AGENTS AND ARTIFICIAL INTELLIGENCE

4.1 INFRARED SENSOR

4.1.1 INFRARED BASED WANDER ALGORITHM FOR ONE AGENT

Using the build-in InfraRed sensors of the Rovio which are placed below its "head", it is easy to create a simple and robust algorithm of wandering around the room avoiding obstacles. This behaviour can also be disrupted from the presence of a human in the room. Therefore, the status of the IR sensor is constantly requested. When the sensor detects an obstacle, the agent stops and rotates searching for a way without obstacles, when it finds a clear view it continues moving forward (Figure 4-1). Finally, Rovio's IR sensor senses objects from the distance of approximately 1 meter.

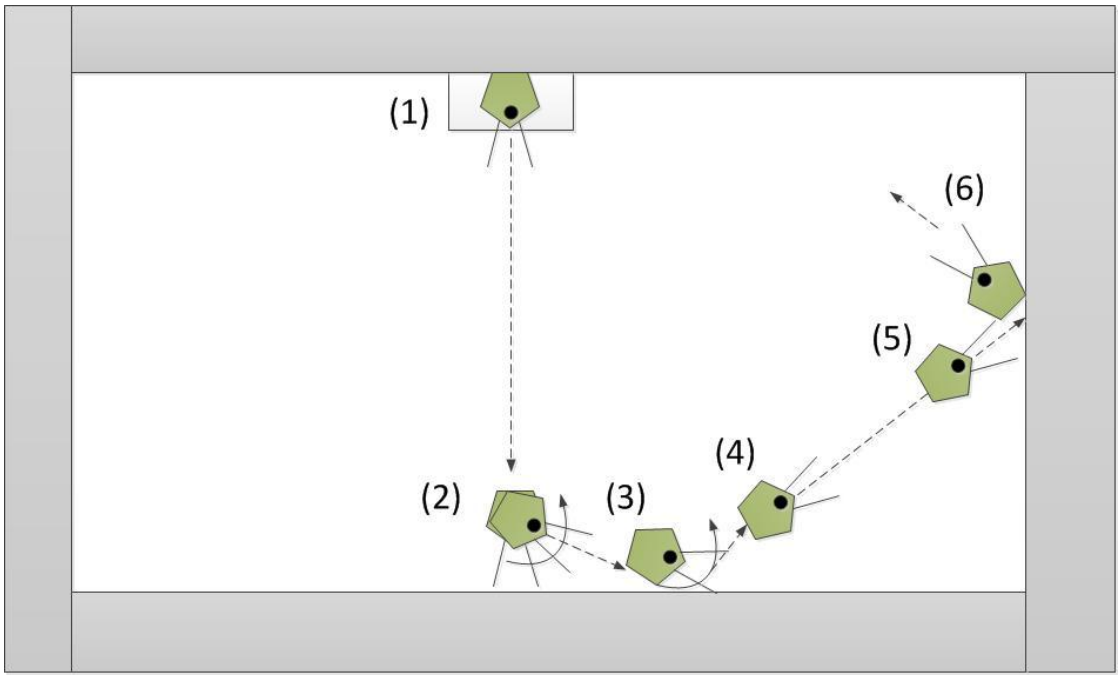

Figure 4-1 InfraRed based simple wander algorithm

The algorithm in pseudocode is as follows:

Wander_Rovio()

1. Read status of IR detector
2. if IR doesn't detect object
3. move forward
4. else

5. rotate anti-clockwise for 10 degrees

This function is inside an infinite while (true) loop.

4.1.2 INFRARED BASED WANDER ALGORITHM FOR TWO ROVIOS

In addition to the simple wander algorithm for one Rovio, a wander algorithm for two Rovios has been created. The simplicity of this algorithm is again the main characteristic. The only difference is that Rovio 2 is programmed to turn clockwise as Figure 4-2 shows. Rovio's IR sensors do not detect the black and blue (from the LEDs) color of the robot, therefore it is convenient to attach white paper or something colourful in the sides of the Rovio's body.

The logic in pseudocode is shown below:

Wander_Rovios()

1. Read status of IR detector for Rovio 1
2. Read status of IR detector for Rovio 2
3. if IR of Rovio 1 doesn't detect object
4. move forward Rovio 1
5. else
6. rotate Rovio1 anti-clockwise for 10 degrees
7. if IR of Rovio 2 doesn't detect object
8. move forward Rovio 2
9. else
10. rotate Rovio 2 clockwise for 10 degrees

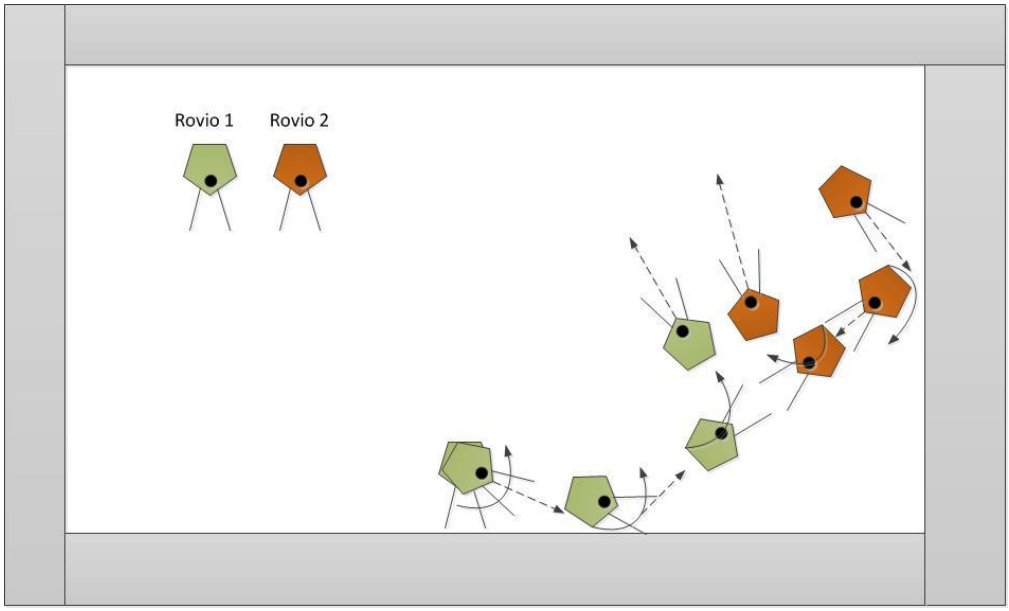

Figure 4-2 IR based wander algorithm for two Rovios

4.2 Digital Image Processing

4.2.1 Color Formats

Color image processing is divided in into two major areas: full-color and pseudo-color processing. In the first category, the images are acquired with a full color sensor. In the second category, the problem is one of assigning a color to a monochrome intensity or range of intensities. Until recently, most digital color image processing was done at the pseudo-color level. However, in the past decade, color sensors and hardware for processing color images have become available at reasonable prices. The result is that full-color image processing techniques are now used in a broad range of applications, including publishing, visualisation and the Internet (Gonzalez, et al., 2002).

Color is the most important characteristic of an image. For humans the recognition and the interpretation of colors are effortless. However, it is not the same for the machines. Therefore, color formats had to be introduced for the representation of the colors. Also, the effect of the different color spaces on the performance of tracking and in general image processing algorithms varies. The main color spaces that are being used are grayscale, RGB and HSV. HSV gives more accurate and more robust tracking results compared to grayscale and RGB images (Comley, et al., 2008).

Grayscale images can be classified as intensity type images where the data that is used to represent the image is a measurement of the intensity or the amount of light. The number of bits used for each pixel determines the number of brightness levels that the pixel will have (Figure 4-3).

A color image can be classified as a set of multiple layers of grayscale images where each layer of the image corresponds to a certain band in the visible light spectrum (Umbaugh, 1998). The information that is stored in each layer of the color image is the brightness in a specific spectral band. The most common used spectral bands are red (R), green (G) and blue (B), the three primary colors in the visible range of the electromagnetic spectrum (Figure 4-3). The RGB color bands are chosen as they correspond to the absorption characteristics of the human eye (Gonzalez, et al., 2002) (Comley, et al., 2008).

While RGB may be the best representation for many applications, like television, it suffers from a number of serious limitations when it comes to activities such as computer based surveillance systems. The main limitations, of the RGB color space is due to the fact that the luminance information that is embedded into each layer of the image. Varying levels of brightness in an image causes RGB values to shift and that introduces instability in the image. The susceptibility of the RGB color space to brightness levels indicates that each layer is equally affected and that the layers are correlated to each other. To overcome this problem, the RGB color space can be normalised to obtain the chromatically information for more robust tracking. It has to be noted that the

chromaticity information obtained from normalising the RGB data is still based on the RGB color space and so is the still easily affected by uneven illumination (Comley, et al., 2008).

In order to overcome the limitation due to variations in the brightness, the RGB color space can be transformed into a different format that decouples the brightness information from the colour information (Umbaugh, 1998). The color spaces have the brightness information separated from the color information. They have one layer of brightness information and two layers of color information. The color spaces that are typically used in video tracking and surveillance are YCbCr and HSV. The implemented methods are using the HSV color format.

The HSV color format has the luminance information in the V layer and chromaticity is placed in the H and S layers (Figure 4-3). The separation of the brightness information from the chrominance and the chromaticity in the HSV color space reduces the effect of the uneven illumination of the image. The utilization of the chrominance and the chromaticity information obtained from the HSV representation enables more robust tracking algorithms to be developed than is possible based on the grayscale and RGB color format (Comley, et al., 2008).

Figure 4-3 RGB, Grayscale and HSV color spaces

4.2.2 Image Processing Methods

This chapter describes the main image processing methods that were used in order to complete simple tasks. These simple tasks were used as a basis to implement more complex ones.

4.2.2.1 Edge Detection

Edge detection methods are used as a first step in line detection processes. Edge detection is also used to find complex object boundaries by marking potential edge points corresponding to places in an image where rapid changes in brightness occur. After these edge points have been marked, they can be merged to form lines and object outlines (Umbaugh, 1998).

This method uses the Canny edge detection operator (Figure 4-4) (Canny, 1986).

The main steps of the algorithm are shown in pseudocode:

Edge_detection()

1. Initialise images
2. Capture BGR image from camera
3. Convert BGR image to GRAY
4. Down-scale the GRAY image
5. Up-scale the down-scaled image
6. Perform the Canny edge detection algorithm to the last up-scaled image

Figure 4-4 Edge Detection using Canny Operator

4.2.2.2 COLOR TRACKING

Color tracking is one of the basic detections. The agents will try to identify objects and by identifying these objects they will behave accordingly. The robustness of the color tracking algorithm is a crucial factor on the success of the implementations. The color detection requires an image where the desired color range is being searched. It has been proven that the best color format for this kind of detection is the HSV.

In pseudocode:

Color_Detection()

1. Initialise images and set color range in HSV values
2. Capture BGR image from Camera
3. Convert BGR to HSV image
4. Search in the HSV image for the color range that we had already set
5. Reduce noise by Smoothing the image
6. Keep the pixels which are in the requested range by Thresholding the image
7. Result in a binary image (white pixels: the pixels in the range)
8. Erode the image
9. Dilate the image
10. Dilate the image

11. Erode the image

Figure 4-6 shows the image processing procedure for the yellow color detection. The colors inside this range are being searched:

HSV minValue : $\{H_{min},S_{min},V_{min}\}$: {25, 180, 180}

HSV maxValue : $\{H_{max},S_{max},V_{max}\}$:{60, 255, 255}

Figure 4-5 shows colors of the previous range in HSV format. The colors range from light brown to light green. However, there are many combinations of the three HSV values, so when the HSV parameters are equal to {43,255,255}, they represent the yellow.

Figure 4-5 Yellow color range in HSV. From left to right, {25, 180, 180}, {30, 189, 189}, {35, 198, 198}, {40, 207, 207}, {45, 216, 216}, {50, 255, 255}, {60, 255, 255} and {43, 255, 255}.

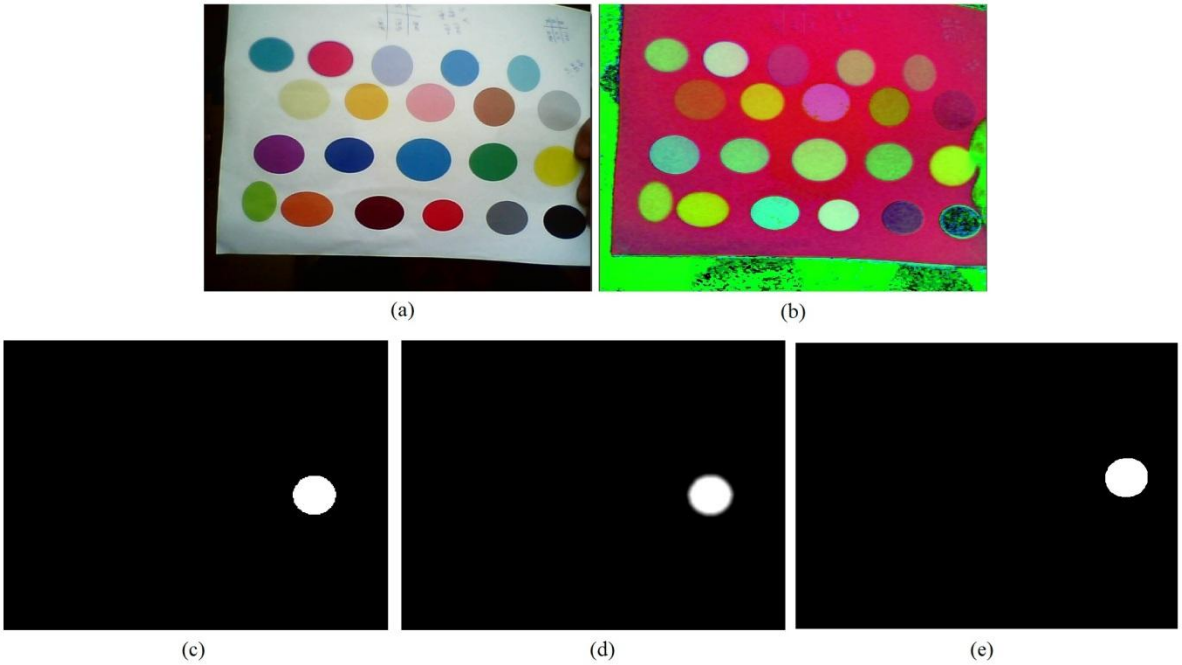

Figure 4-6 Yellow Color Tracking. (a) BGR image, (b) HSV image, (c) Search color range in HSV, (d) Smoothed, (e) Thresholded with Opening (erode-dilate) and Closing (dilate-erode)

Narrowing the color range results to more specific color detection. The previous search for yellow can detect colors from light brown to light green. The range of the HSV values was very large. For Hue was: 60-25=35, for Saturation was: 255-180=75 and the same range was for Value. By careful testing of the algorithm the appropriate HSV parameters can be set for the search of a specific color. By narrowing the range of the H, S and V values higher accuracy can be succeeded. With the following values different shades of blue can be detected (as shown in Figure 4-7):

HSV minValue : {H_{min},S_{min},V_{min}} : {110,190,0}

HSV maxValue : {H_{max},S_{max},V_{max}} {120, 225, 255}

Figure 4-7 has two blue spots of different shade. However, by setting properly the input search range the method detects the darker one. That happened mainly because of the very narrow range that has been chosen in H and S space. The first two parameters, H and S, have color information while the third, V, represents the intensity. This last parameter, V, is not so crucial for the color tracking, as far as it is in reasonable range. In comparison with the yellow HSV range, H ranges just for 10 and the S only for 35.

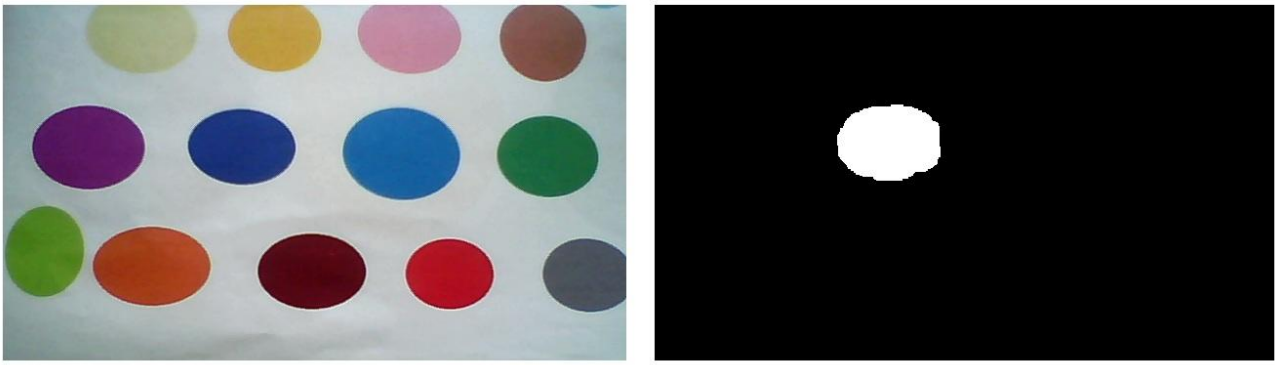

Figure 4-7 Blue Color Tracking. RGB and Binary image

4.2.2.3 PINK BALL DETECTION

One of the first tasks was to make Rovios interact with a pink ball. Therefore, an algorithm was needed for detecting the ball. The method that was implemented focused on the color and not on the shape of the ball. This method also searches in two color ranges, and that is because pink/red color exists in both sides of the HS palette (Figure 4-8). Furthermore, another characteristic that is very useful in the developed AI methods is the display of the mass centre of the detected pink/red area.

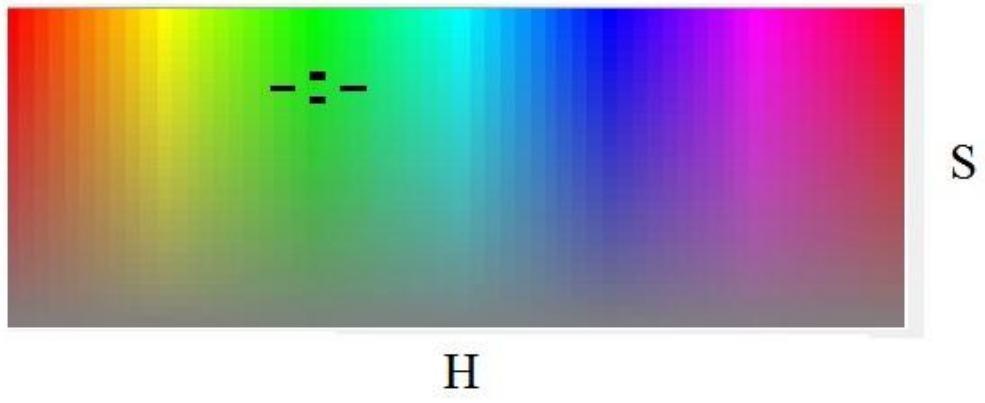

Figure 4-8 HS palette

Pink_Ball_Detection()

1. Initialize images
2. Set the upper and lower color range
3. Capture BGR image from the camera
4. Convert BGR to HSV
5. Search in HSV image for the lower range (thresholded image 1)
6. Search in HSV image for the upper range (thresholded image 2)
7. Perform logical OR between the two thresholded images (result in one thresholded image)
8. Reduce noise by Smoothing the last thresholded image
9. Keep the pixels which are in the requested range by Thresholding the image
10. Result in a binary image (white pixels: the pixels in the range)
11. Erode the image
12. Dilate the image
13. Dilate the image
14. Erode the image
15. Calculate mass point
16. Display mass point

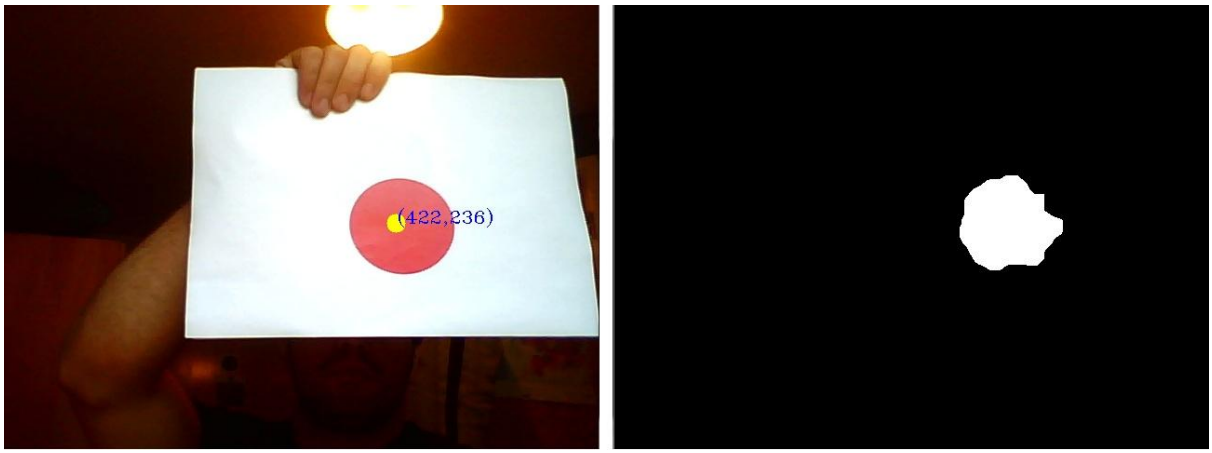

Figure 4-9 Pink Color Detection and mass centre display

4.2.2.4 SEGMENTING THE IMAGE

The previous method calculates and displays the mass centre of the pink ball. Theagent has to search, detect and move towards the ball. The position of the mass centre of the ball on the image denotes the relative distance from the agent. Therefore, the position of the ball (mass centre) is

displayed on the image and commands are generated automatically based on the current position. The idea is based on the segmentation of the image in rectangles and the display of the detected mass centre.

The main logic is depicted in the pseudocode below:

Segmenting_Image()

1. Initialise the number of segments (e.g. n, total segments=nxn)
2. Initialise rectangle color
3. Initialise points color
4. Capture BGR image from camera
5. Create n+1 points in y dimension with a distance height/n
6. Create n+1 points in x dimension with a distance width/n
7. Display the coordinates of each point
8. Draw rectangles based on the points

Figure 4-10 Segmenting for n=4 (nxn=16 segments) and n=8 (nxn=64 segments)

4.2.2.5 Dynamic Color Range Adjustment

With the above algorithms implemented the Rovios are able to detect and play with a pink/red ball in their environment. But what will happen if we replace the pink ball with a ball of another color?

The answer is that the Rovios have to detect the ball (circle detection), sample its color and then set a new color range which will be based on the color that they found. The color detection is faster and more robust than the circle detection algorithm in a dynamic and complex environment. Apart from that, Rovios are not able to find the pink ball (detect the color) in different light conditions. With such an algorithm the robots will be able to track the ball and set dynamically the new color

range that they have to search for. With this algorithm the agents will behave autonomously and dynamically inside a dynamic environment.

The pseudocode below describes this method:

Dynamic_Color_Range_Adjustment()

```
1. Initialise images
2. Initialise color of the centre and the inner and utter circumference of the detected circle
3. while (true)
4.     Capture RGB image from Camera
5.     Convert image to Grayscale
6.     Apply threshold to the Grayscale image
7.     Smooth the resulted binary image
8.     Apply Canny edge operator
9.     Search for circles using the Circular Hough operator
10.    Set the number of circles to be displayed/detected
11.    Display centre and the inner and utter line of their circumference
12.    Print in console this information
13.    if ('a' button pressed)
14.        Search for the centre of the circle
15.        Take the color of four neighbour pixels in RGB
16.        Display in the Console this RGB color
17.        Calculate their average RGB value and display it in the Console
18.        Convert the last average value of RGB to HSV and display it in the Console
19.    end if
20.    if ('c' button is pressed)
21.        Set the value range of the HSV parameters
22.        Check them to be in range (0,255)
23.        Set the HSV range dynamically based on the last average value of HSV
24.        Initialize images and appropriate secondary variables and structures
25.        while (true)
26.            Capture RGB image from Camera
27.            Convert RGB to HSV image
28.            Search the HSV image for the new dynamically set color range
29.            Smooth the image
30.            Apply threshold
```

```
31.                 Erode the Binary image
32.                 Dilate the image
33.                 Dilate the image
34.                 Calculate the mass centre of the blob
35.                 Display the mass centre and its coordinates
36.                 if (‘ESC’ pressed)
37.                         break while loop
38.       end if
39. if ((‘ESC’ pressed)
40.       break while loop
```

As the pseudocode clearly depicts, in the first part of execution of this algorithm the Circular Hough Operator has been used to detect the circles. The parameters are set up to detect only one circle. In this case the experimental environment has to have only one circle somewhere otherwise the method won’t be stable. Figure 4-11 and Figure 4-12 shows this first step of circle detection and real-time display of the centre and its circumference (graphical and in console).

Figure 4-11 Circle Detection

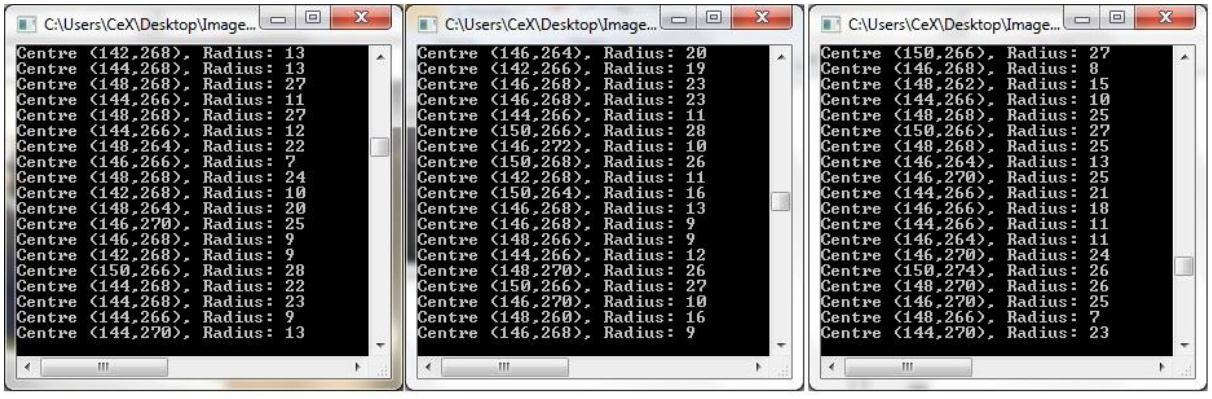

Figure 4-12 Centre and Radius display on Console

As the pseudocode describes when the detected area is stable around a circular shape, button “a” has to be pressed in order to take the values of four neighbouring pixels of the centre of the detected circle. Figure 4-13 shows the sampling of the values in RGB. First the four values in RGB and their

average is displayed. The average is calculated among all the button pressings and as a result longer sampling period (times of pressing ‘a’) means more accurate color calculation. After the sampling procedure the last average value of RGB is converted into HSV format.

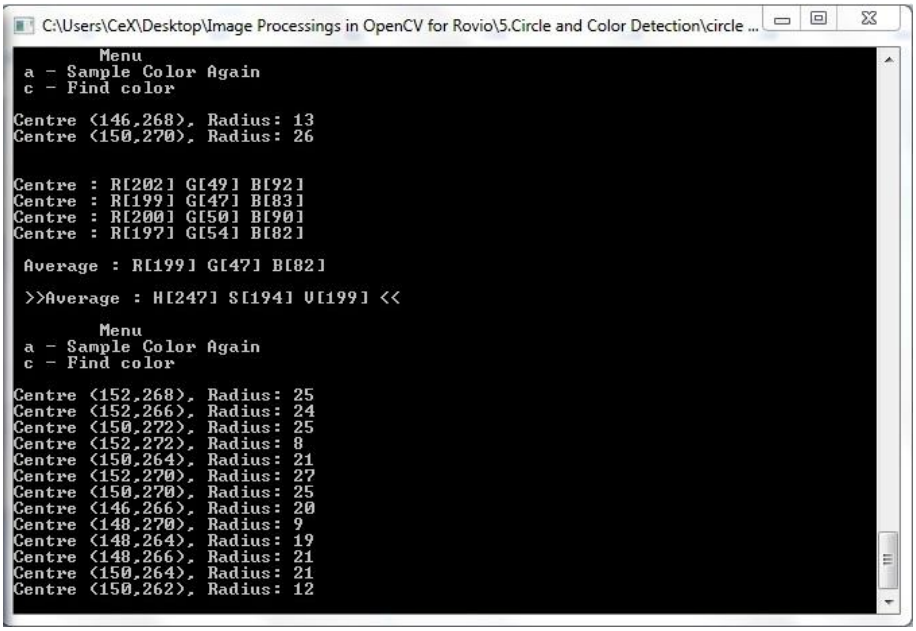

Figure 4-13 Sampling and Averaging neighbour pixels of the centre

Figure 4-14 shows the setup of the HSV range based on the last average HSV values. Inside the code there are the variables which set up the new range. These variables in this example are:

Hue range: 180

Saturation range: 15

Value range: 15

The values must be between 0 and 255. Therefore in case that the values exceed these borders the algorithm sets them 0 and 255 for the lowest and the highest value respectively.

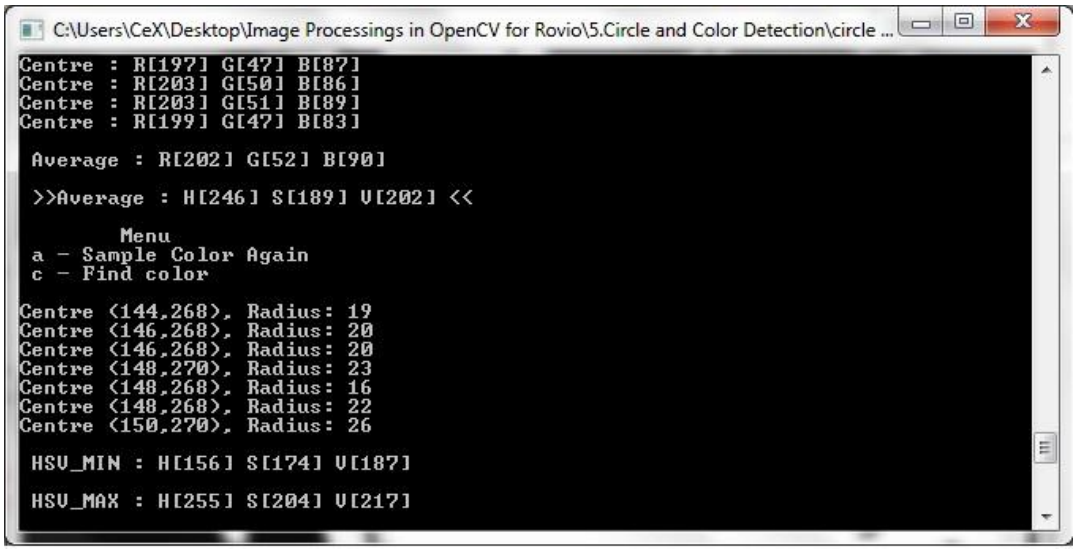

Figure 4-14 Last Average and Color range adjustment in HSV

By pressing the ‘c’ button, the function for searching in this range is being enabled. Figure 4-15 shows the detected area in the binary image and the display of the mass centre.

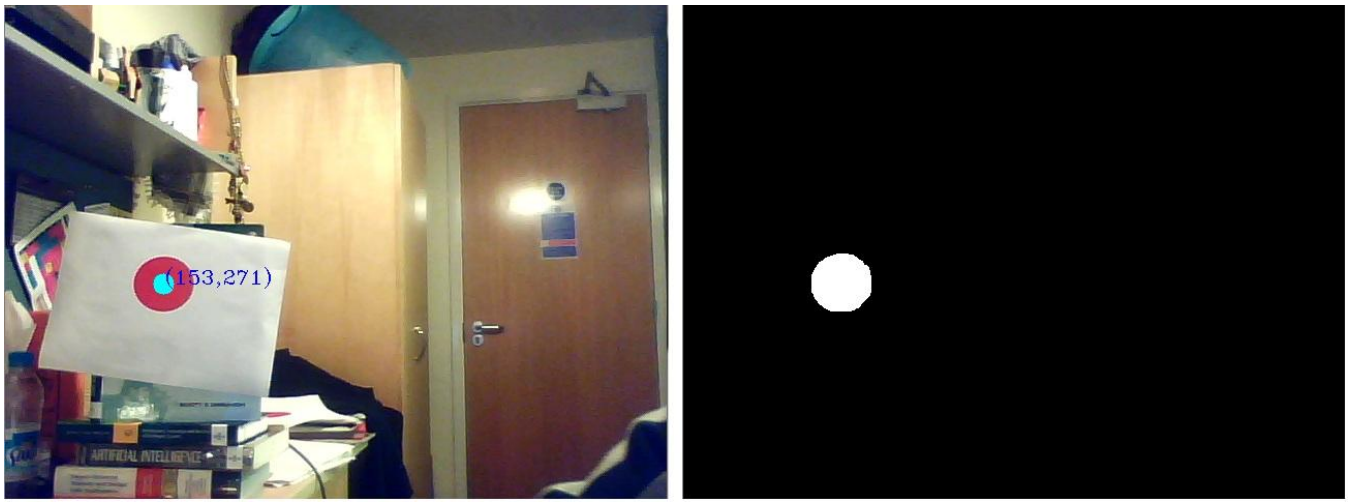

Figure 4-15 Dynamic color detection and mass centre display

4.3 Search And Find Pink Ball

After implementing the previous image processing algorithms Rovios are able to search, find and approach the ball (combining the pink color tracking and the image segmentation). Rovios can autonomously wander around the room based on their vision on their search for the ball.

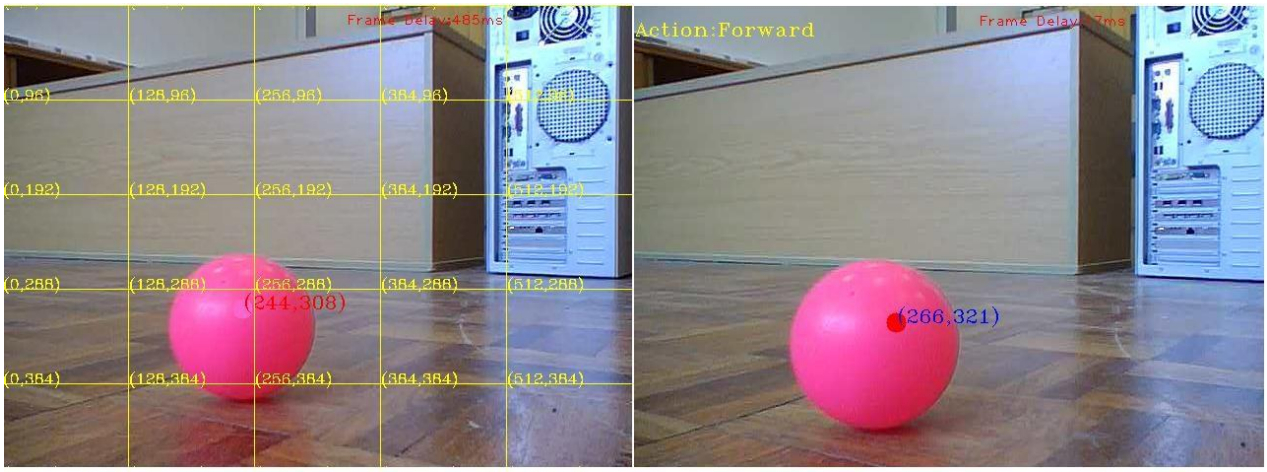

Figure 4-16 Detecting and Commanding

Figure 4-16 shows two images taken by Rovio’s camera. The image on the left shows how Rovio detects the ball and displays the mass centre on the segmented image. The image is segmented into 5x5=25 rectangular areas. The command that is given to Rovio depends on the actual position of the mass centre on the segmented image. The command is displayed on the top left corner of the right image.

The pseudocode below depicts the logic for approaching the ball:

```
Auto_Commanding()
 1. Calculate mass centre of the pink ball
 2. if (centre.y coordinate > 400)   //very low of the image
 3.      stop Rovio
 4.      display Stop Command
 5. else if (centre.y coordinate > 96)
 6.      if (centre.x coordinate > 512)   //far right of the image
 7.              move forward right
 8.              display command
 9.      else if (centre.x coordinate > 348)   //centre right
10.              move forward
11.              display command
12.      else if (centre.x coordinates > 256)          //centre of the image
13.              move forward
14.              display command
15.      else if  (centre.x coordinates > 128)         //centre left
16.              move forward
17.              display command
18.      else                                  //far left of the image
19.              move forward left
20.              display command
21. else
22.      rotate left (anti-clockwise) appr.10 degrees  (basically, rotate and stop together)
23.      display command
```

Figure 4-17 depicts Rovio playing with the ball. Rovio starts from position 1, there is no pink ball on its sight, so it rotates anti-clockwise searching for the ball (2). Finally, in position (3) detects the ball and moves toward it according to the Auto_Commanding. Rovio finds and kick the ball (4). Then, the ball bounces (5) and Rovio searching again for it (6). When the ball is again on its sight (7), moves and kicks it again (8).

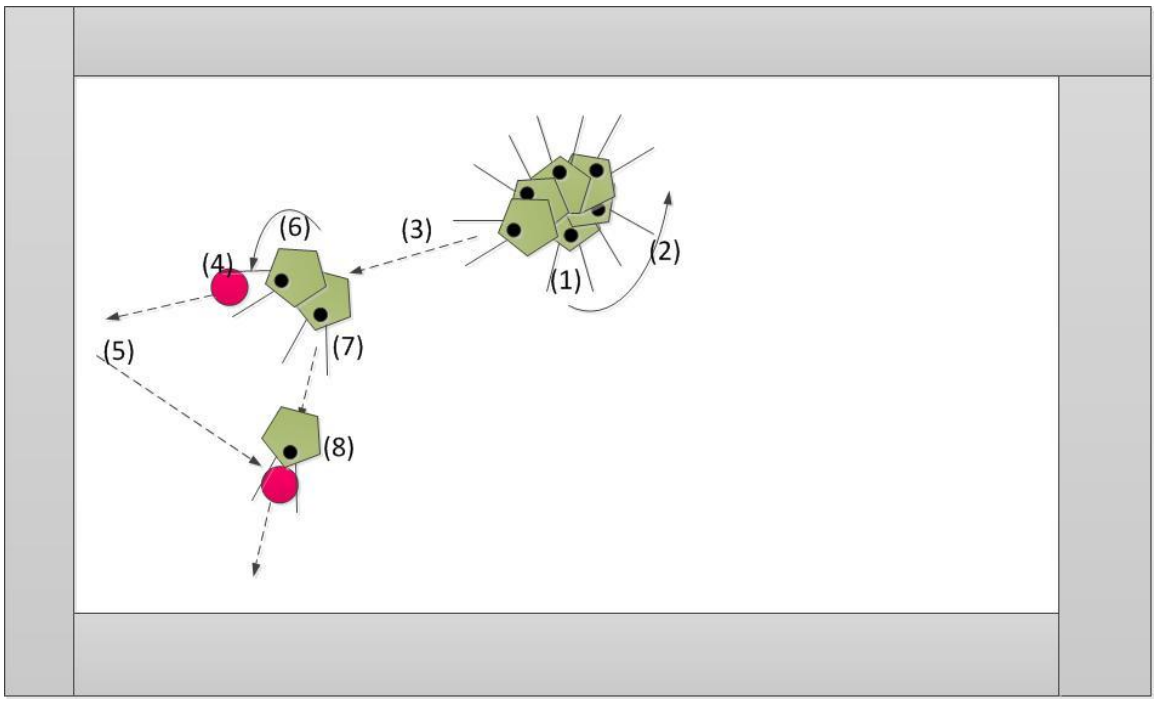

Figure 4-17 Rovio playing with the ball

The same method has been implemented for a yellow ball. The only difference is the input range of color (instead of pink, a yellow color range is inputted).

4.4 ROVIOS BALL PLAY

A simple scenario which involves two Rovios has been created after the implementation of the previous algorithm/method. Rovio 1 will search, detect and "kick" the ball, after its completion Rovio 2 will do the same, then Rovio 1 again and so on. Figure 4-18 shows the state transition for each Rovio. Rovio moves from Idle when the user gives the command to execute this scenario. When Rovio 1 finishes, gives permission to Rovio 2 to start. The same happens when Rovio 2 finishes.

The agents change their environment by changing the position of the ball. The position of the ball is not fixed.

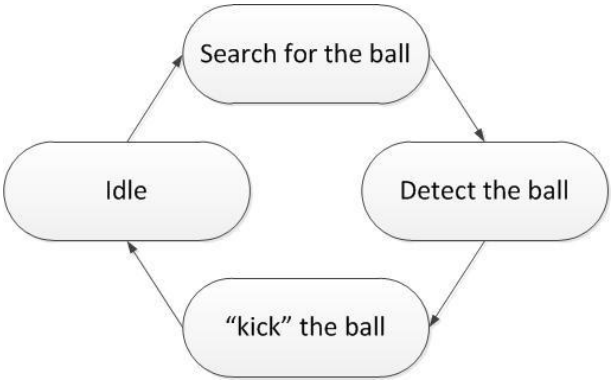

Figure 4-18 State transition for each Rovio

4.5 VISUAL SERVOING AND NAVIGATION SCENARIO

After the successful implementation of the previous algorithms/methods and especially with the help of the two methods which detect and approach the pink and the yellow ball, a servoing and navigation scenario was created. In this scenario, Rovio is beginning from its base and searches for the pink ball. When it gets close to the pink ball, the goal changes and it searches for the yellow ball. After finding the yellow ball, Rovio is again searching for the pink ball which possibly changed position because of the first touch. After finding the pink ball for the second time Rovio heads to its home position. This scenario is depicted in Figure 4-20 (the numbers show the sequence of the actions).

The sequence of these actions can be depicted in the block diagram (Figure 4-19). The user is enabling this scenario by simply pressing the appropriate button on the GUI. The GUI enables an automated and autonomous behaviour while the numbers in the figure below describe the sequence of events or states that the robot is passing through.

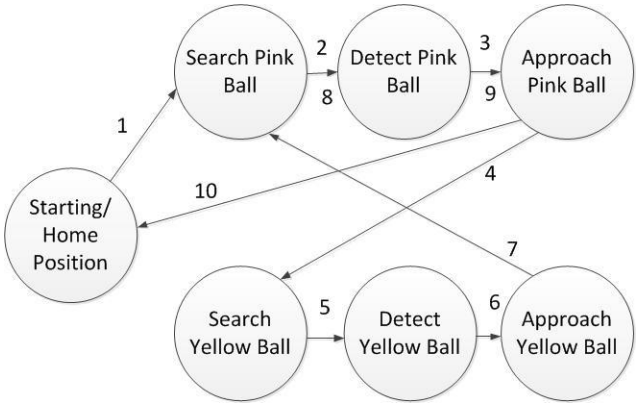

Figure 4-19 Sequence of actions for completing Scenario

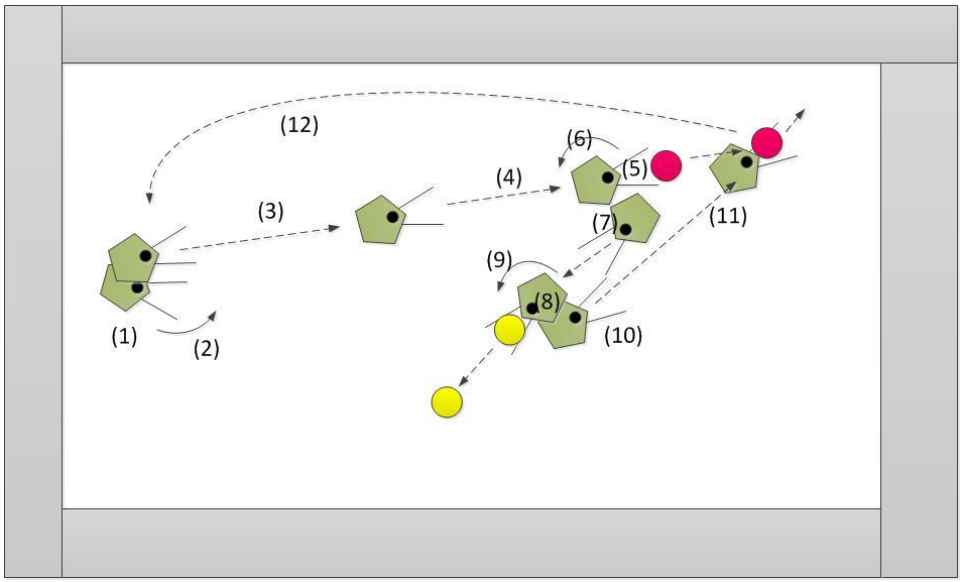

Figure 4-20 Navigation result for Scenario

4.6 POSITIONING SYSTEM

The system consists of three values: x and y coordinates and theta value. Figure 4-21 depicts the logic of the positioning system in the room. The red lines represent the two infrared beams. The positioning system is based on their position.

The build-in positioning system of the Rovios is very unstable. The coordinates are being read from the status string.

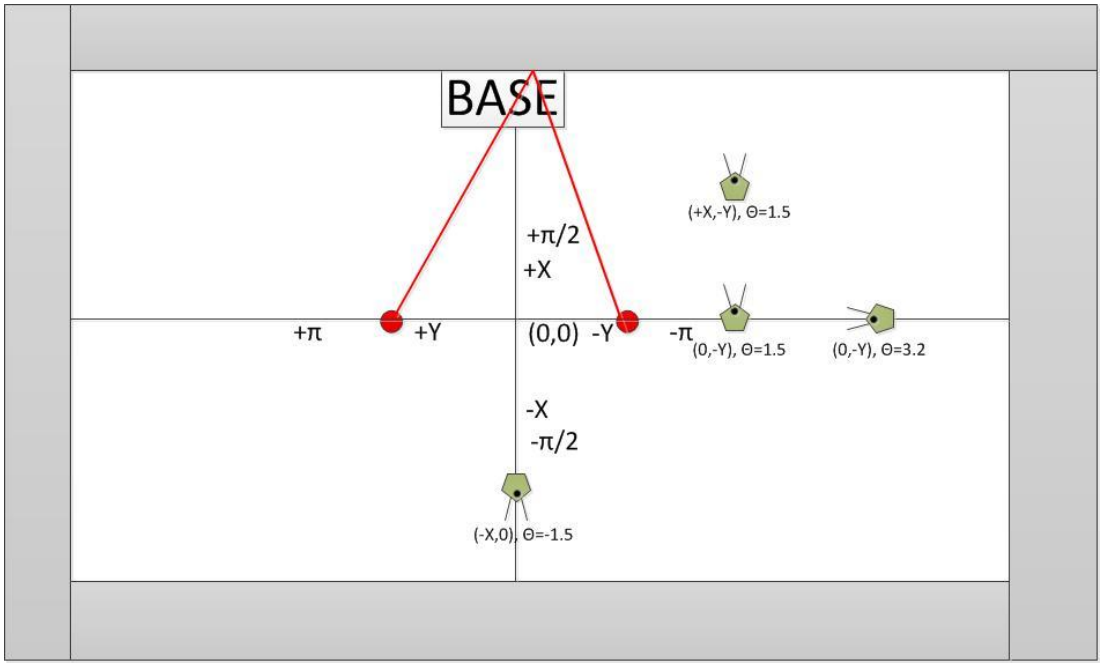

Figure 4-21 Positioning

4.7 SUMMARY

After describing the logic of the developed methods, the following chapter focuses on the way that these methods had been implemented describing the tools and the techniques that were used.

5 IMPLEMENTATION

The software implementation of the current project is a Windows Form Application build in Visual Studio 2008 in Visual C#. Rovio's C# API is being incorporated in this software project.

All the code is included in Appendix A and B.

5.1 OPEN.CV AND EMGU.CV

Open CV libraries are used to implement the image processing methods. Open CV is an open source computer vision. The library is written in C and C++ and runs under Linux, Windows and Mac OS X.

Open CV was designed for computational efficiency and with a strong focus on real time applications. Open CV is written in optimised C and can take advantage of multicore processors. The library contains more than 500 functions and one of its goals is to provide a simple to use computer vision infrastructure that helps developers build fairly sophisticated vision applications quickly (Bradski, et al., 2008).

Open CV was helpful on the first stages for developing and testing the methods in DevC++ compiler. However, when it came to implement the methods to Rovios this wasn't enough. Therefore, Emgu CV was used which is a cross platform .NET wrapper to the Intel Open CV image processing library. This wrapper allows Open CV functions to be called from .NET compatible languages such as C#, VB, VC++ and IronPython.

5.2 GRAPHICAL USER INTERFACE

This chapter presents and analyses the functionalities of the Graphical User Interface (GUI) that was designed. The GUI has two main tabs and is video-based. The first tab controls two Rovios of our choice (out of the five) and contains all the buttons and controls for the execution of the scenarios and the image processing methods. The second tab manually controls the five Rovios.

5.2.1 TAB 1

Figure 5-1 shows Tab 1. The main characteristics of this interface are the two video streams that are being acquired from current Rovio 1 and 2. In the top of the images, Rovio's current coordinates and theta value are displayed. Additionally, their distance is calculated and displayed.

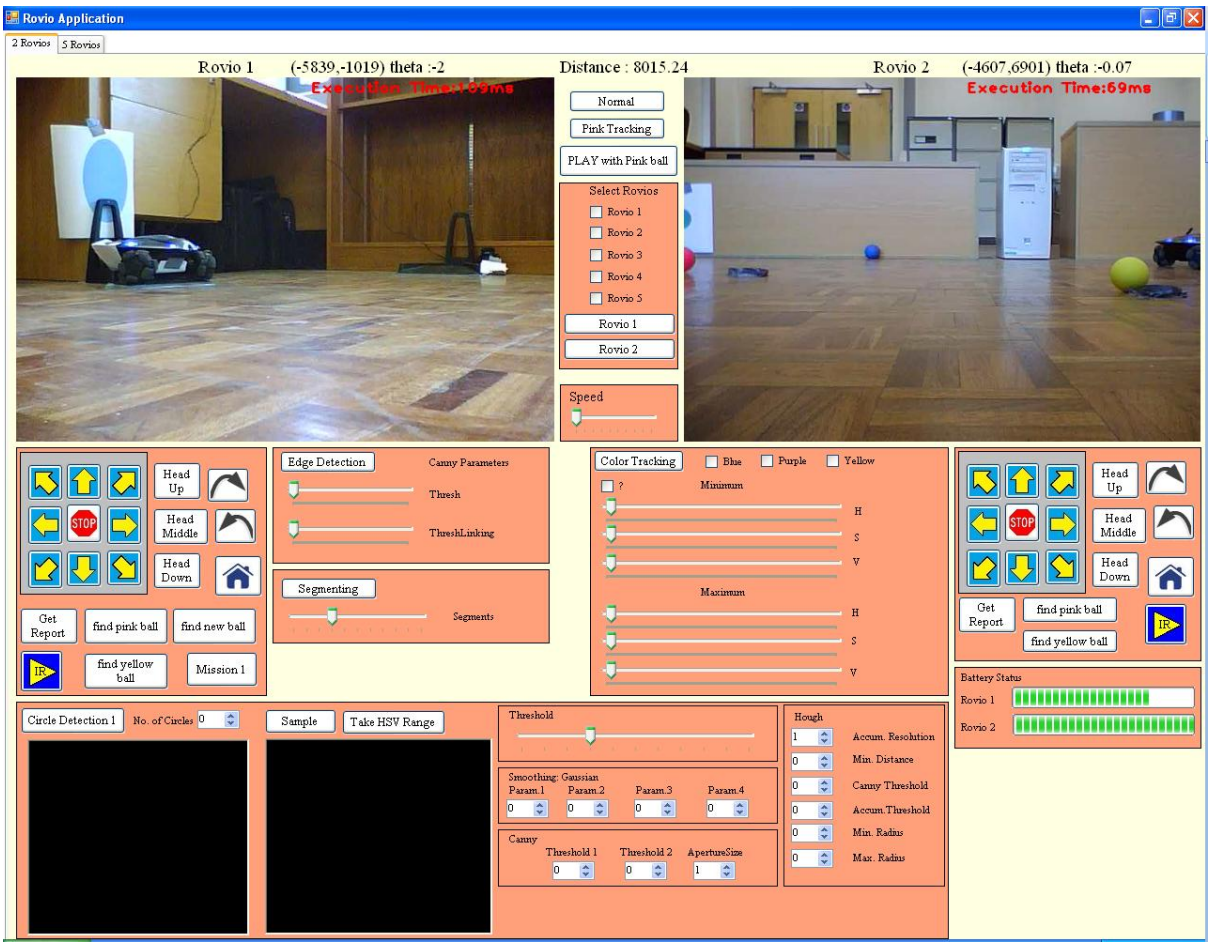

Figure 5-1 Tab 1

5.2.1.1 MANUAL CONTROLS

Under the images the panels for the manual control of the robots have been placed (Figure 5-3). Rovios are omni-directional agents and that is the reason why buttons command movement to all directions (forward, backward, left, right, forward left, forward right, backward left, backward right). Furthermore, the stop button stops the current movement. Next to these controls there are two buttons for rotation (clockwise and anti-clockwise) movements and three buttons that adjust the position of the head. In the same panel, there is the button which commands Rovio to go home and dock to its base. Figure 5-2 shows the state diagram which describes the functionality of the manual controls.

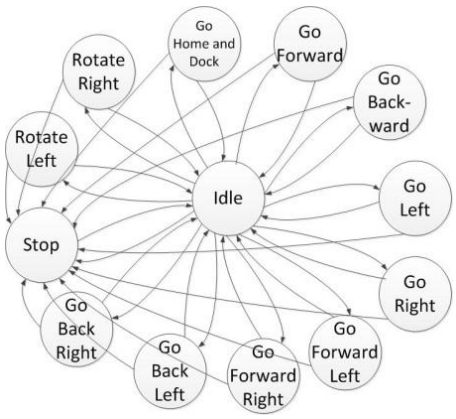

Figure 5-2 Manual Control State Diagram

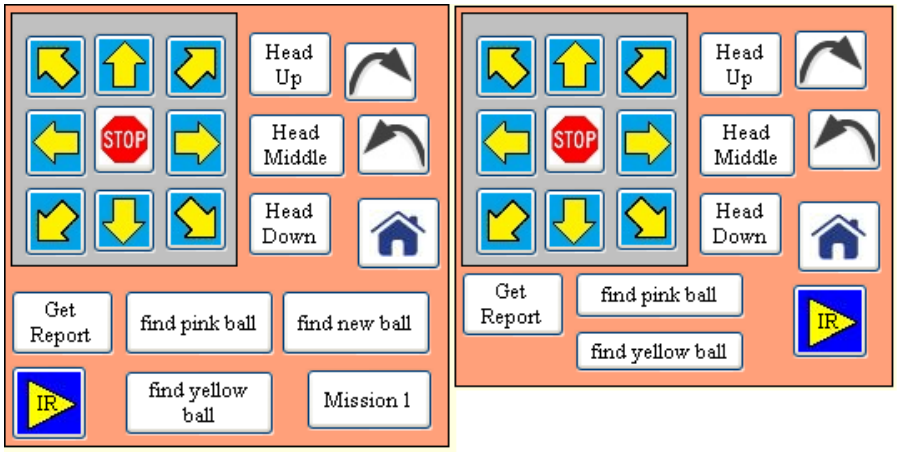

Figure 5-3 Manual Control Panels

5.2.1.2 REPORT STRING

Below the manual control buttons there is the "Get Report" the button which displays in a new window the status of the agent. Figure 5-4 shows the two windows with the status string of the two Rovios. Critical information is being displayed in these windows for the control of the robots. First of all, current coordinates x, y and theta are being displayed. The flags display the status of the IR sensor. After that the settings of the built-in camera are being displayed. The battery level and the charging state are other important information that is being used in this project.

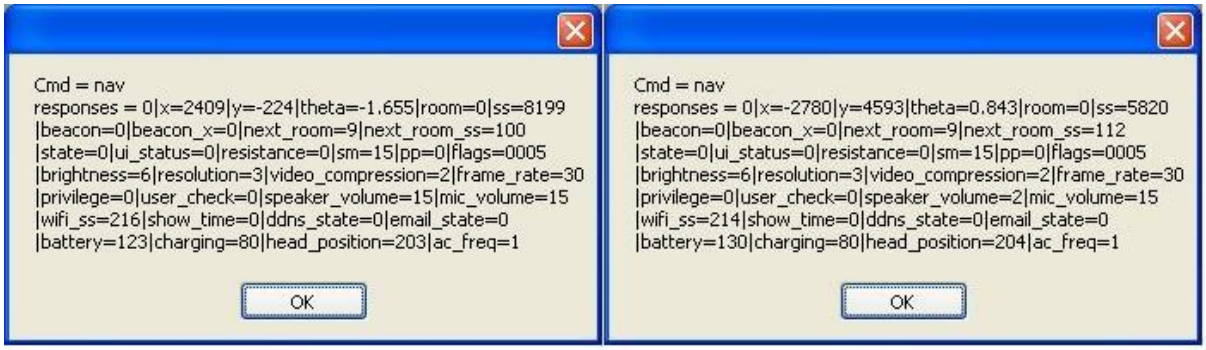

Figure 5-4 Status Reports

The activity diagram in Figure 5-5 depicts the procedure that is followed in order to fetch and display the status string.

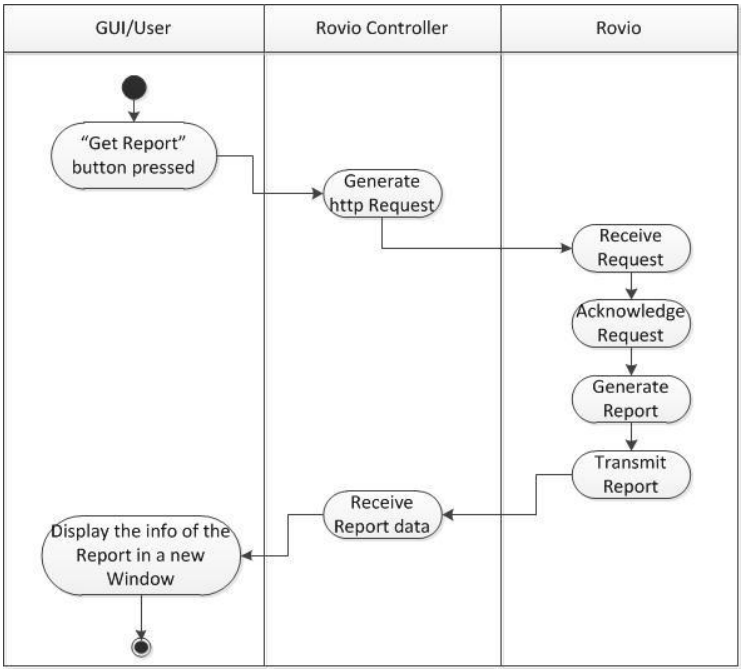

Figure 5-5 Get Report Activity Diagram

5.2.1.3 IR WANDER BUTTONS

The two yellow play/pause buttons enable/disable the IR based wander algorithm for each Rovio as the Activity diagram depicts in Figure 5-6.

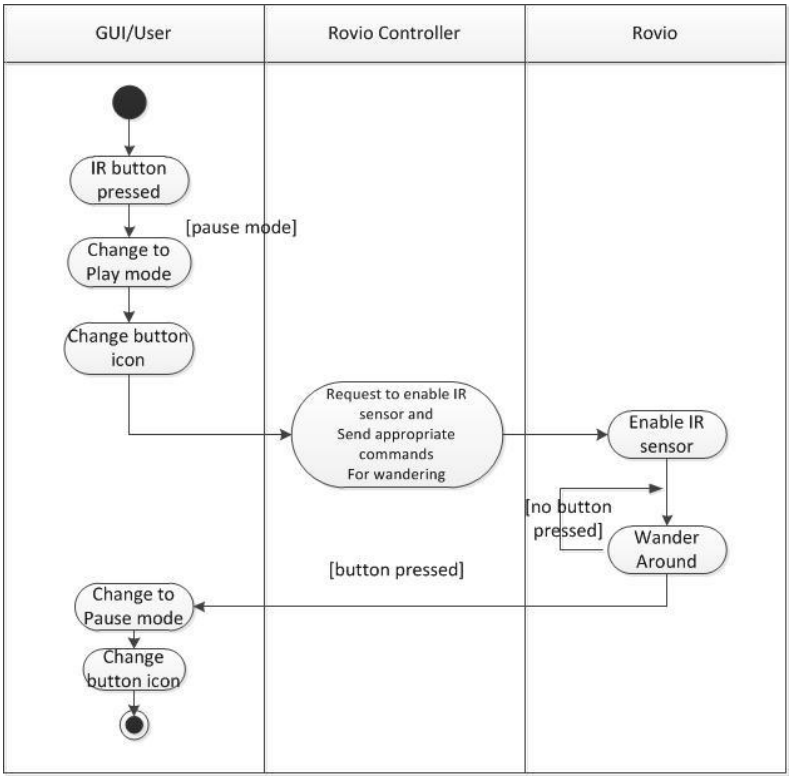

Figure 5-6 IR buttons Activity diagram

5.2.1.4 FIND BALL AND MISSION BUTTONS

The buttons “find pink ball” and “find yellow ball” command Rovios to search, detect and approach the corresponding ball. These buttons search for an already set up color range, in these cases pink and yellow color range. Figure 5-8 depicts the flowchart for these functionalities for current Rovio 1 (something similar has been programmed for current Rovio 2). Furthermore, the “find new ball” button searches for the new dynamically set up color range. Finally, button “Mission 1” enables Rovio to execute the servoing and navigation scenario that was described in 4.5. Scenario makes use of the methods which find and approach the pink and yellow ball and finally drives home (Figure 5-7).

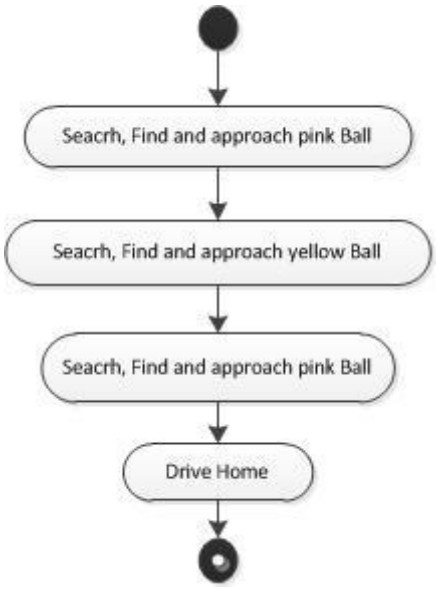

Figure 5-7 “Mission 1” activity diagram

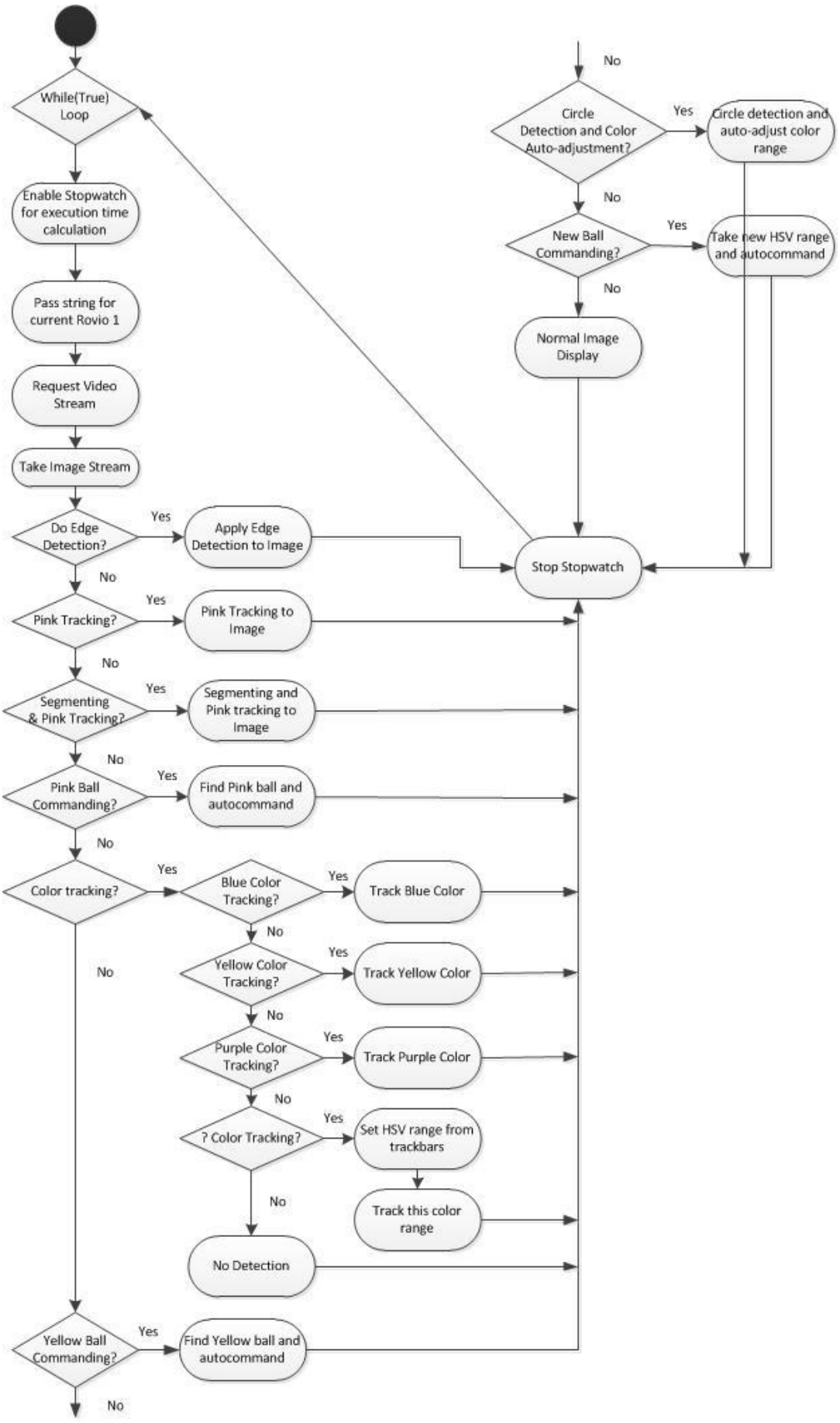

Figure 5-8 Current Rovio 1 button functionalities

5.2.1.5 NORMAL OPERATION BUTTON

The button on the top of the GUI brings the state of the operation and display into normal operation. Normal state of the application is the state where the standard threads are running and any operation can be triggered. As the flowchart shows in Figure 5-8 this is the default operation.

5.2.1.6 PINK BALL TRACKING BUTTON

By pressing the "Pink Tracking" button (on the central top of the GUI), the pink ball is detected and its position is displayed on the processed image (Figure 5-8 and Figure 5-9). The ball is in the same position but two Rovios from different positions can detect it (Figure 5-9).

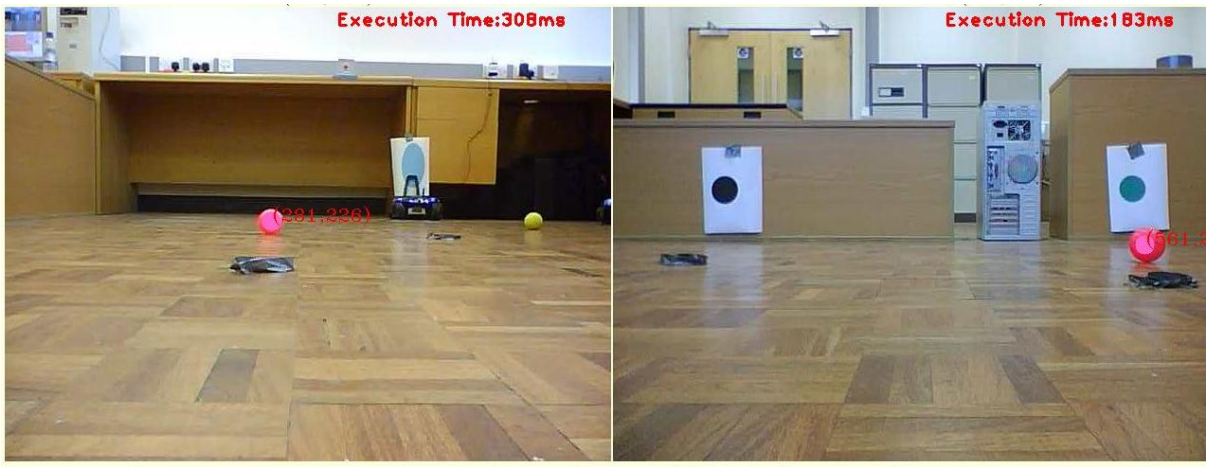

Figure 5-9 Pink Ball tracking

5.2.1.7 PLAY WITH PINK BALL BUTTON

This button enables the scenario for the current two Rovios to play with the pink ball as it is described in 4.4.

5.2.1.8 SELECTING ROVIOS

Figure 5-10 shows the panel where Rovio 1 and 2 can be chosen. By default Rovio 1 and 2 are the Rovios with IPs "http://192.168.2.11" and "http://192.168.2.12" respectively. However, agent swapping can be performed by clicking the desired Rovio and address it as Rovio 1 or 2. Immediately, the incoming image stream will change. This is a very dynamic characteristic as it makes it feasible to change agents during execution time.

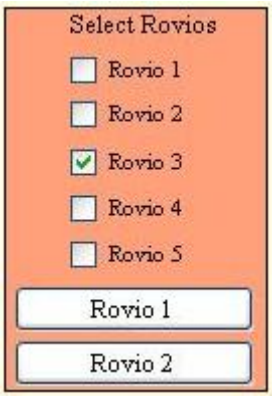

Figure 5-10 Select Rovio 1 and 2

5.2.1.9 SPEED CONTROL

Trackbar speed controls the speed of the robots. At position 1 Rovios have their max speed and in position 10 their minimum. By default they operate in max speed. This functionality was accomplished by setting a public variable which is universally changeable from the trackbar (Figure 5-1).

5.2.1.10 EDGE DETECTION PANEL

Enabling the edge detection, using the Canny operator, the two trackbars can be adjusted for different results (Figure 5-11 and Figure 5-8). The first trackbar adjusts the Threshold while the second one adjusts the Threshold Linking parameter of the Canny algorithm.

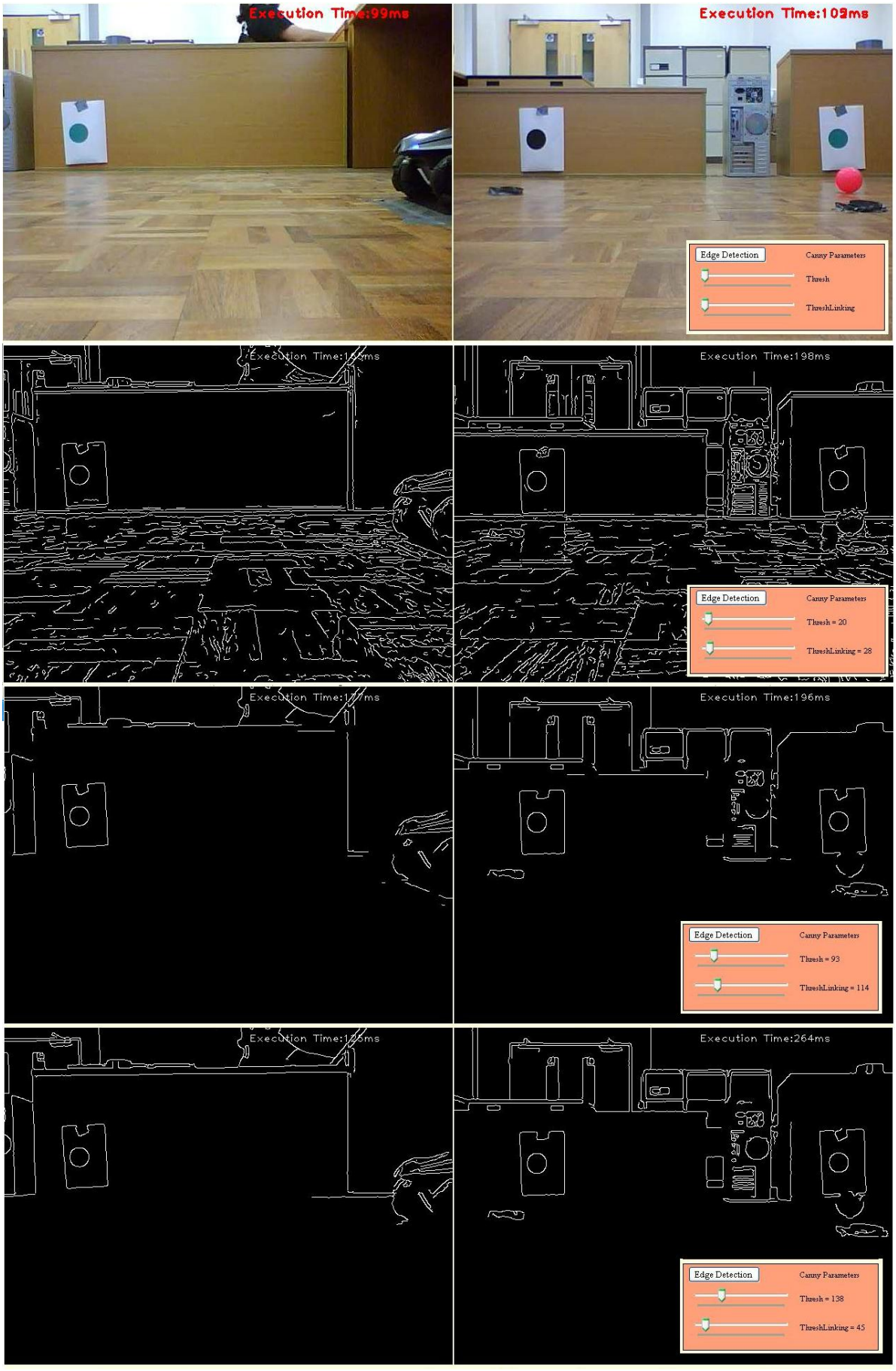

Figure 5-11 Edge Detection panel and manual adjustment

5.2.1.11 SEGMENTING PANEL

Figure 5-12 shows the functionality of button “Segmenting”, and the different result by adjusting the number of the segments while Figure 5-8 shows how this functionality can be enabled. This button segments the screen according to the trackbar and detects and displays the ball’s position at the same time. These images helped to develop the Auto_Commanding() algorithm that was presented in 4.3.

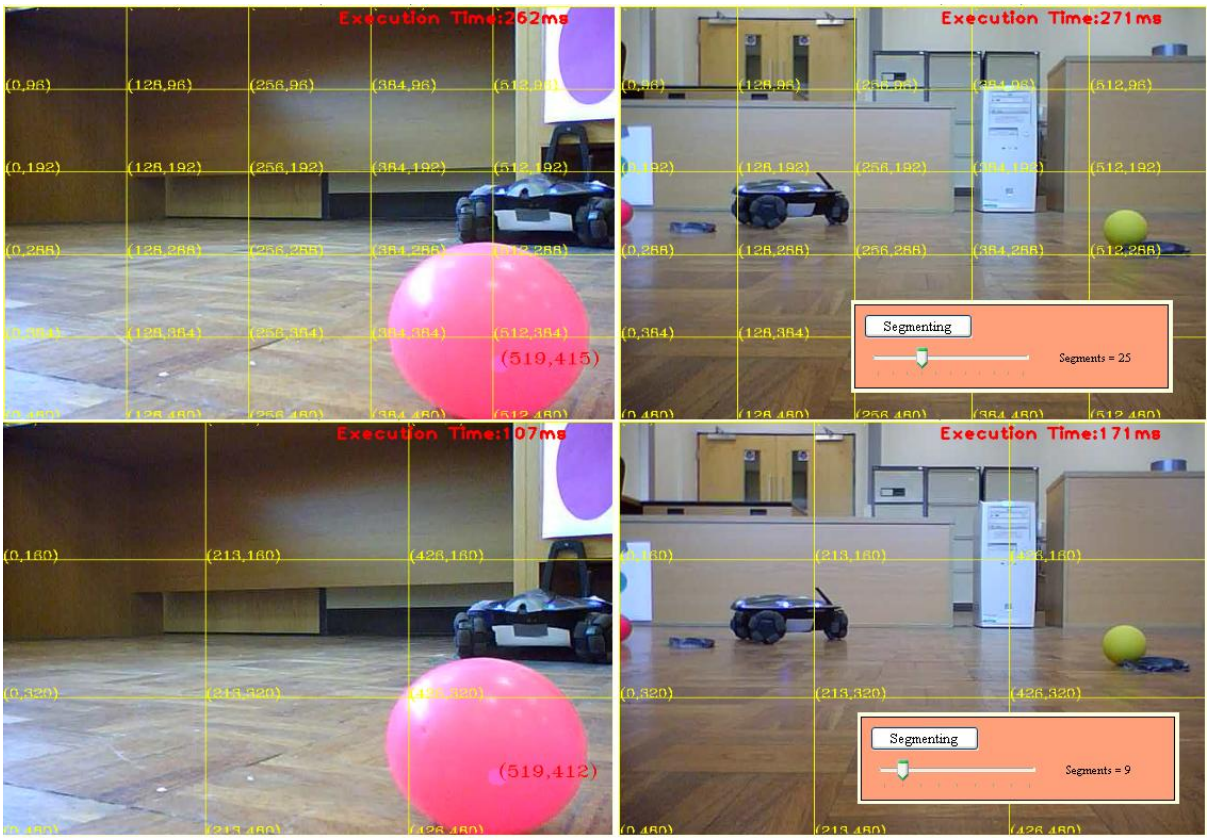

Figure 5-12 Segmentation and trackbar (up: 25 segments, down: 9 segments)

5.2.1.12 BATTERY MONITORING PANEL

Figure 5-13 shows the battery monitoring panel. In case of charging the progress bars are “rolling”. Appropriate number of green lines is being displayed by extracting the level of battery of each Rovio from the status string.

Figure 5-13 Battery Monitoring

5.2.1.13 COLOR TRACKING PANEL

Figure 5-14 shows the color tracking panel. The button “Color Tracking” enables/disables the current tracking. There are four checkboxes. The first three enable a predefined search in these color ranges (Blue, Purple and Yellow). The last checkbox (?) enables the manual mode. In this mode, the desired HSV color range is manually set while the result is displayed on the binary images. The way that this panel works is depicted in the flowchart in Figure 5-8.

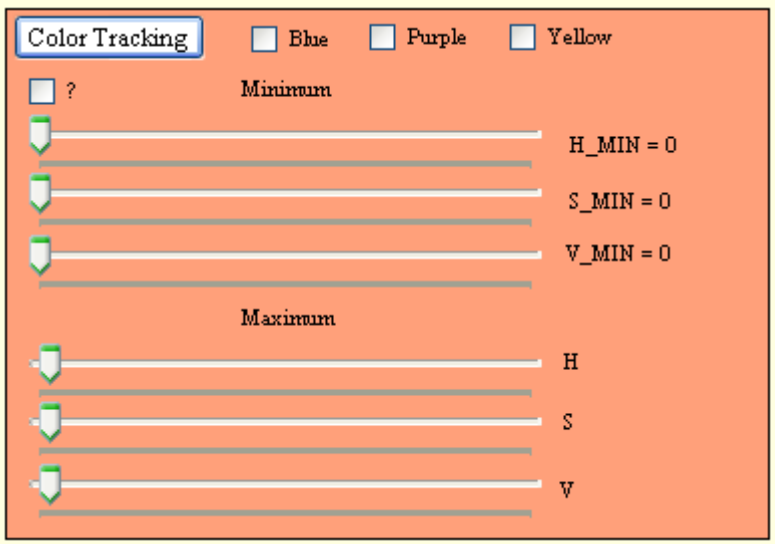

Figure 5-14 Color tracking panel

Figure 5-15 shows the original images from the Rovios (up) and the blue detected areas on these images (down).

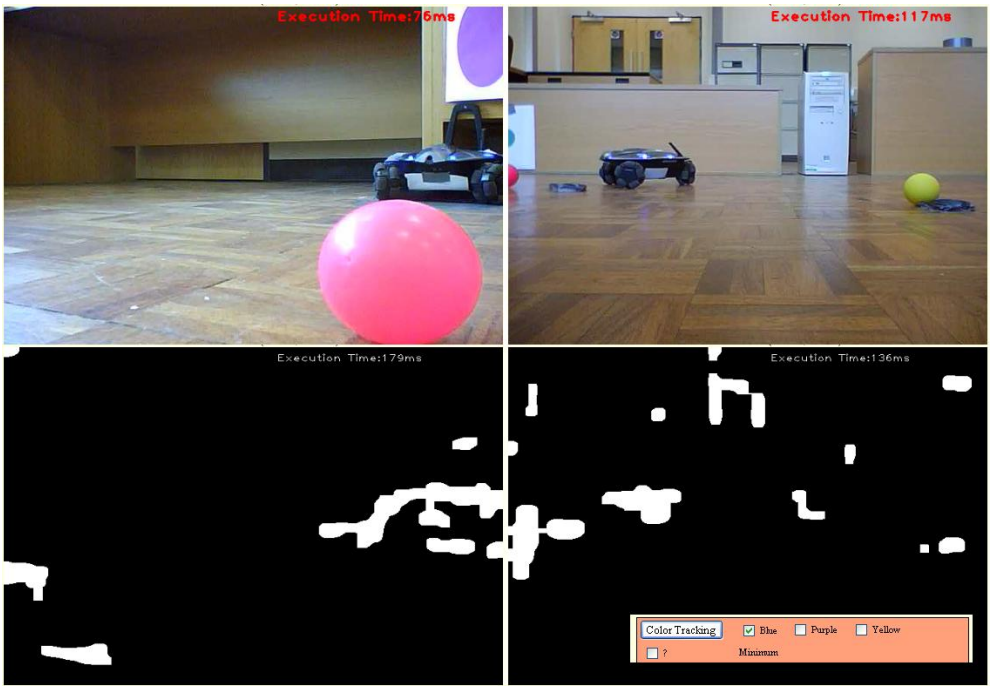

Figure 5-15 Blue Color tracking

In the same way Figure 5-16 shows the result of searching in the yellow color range. The detection of the yellow ball on the second binary image is clearly shown.

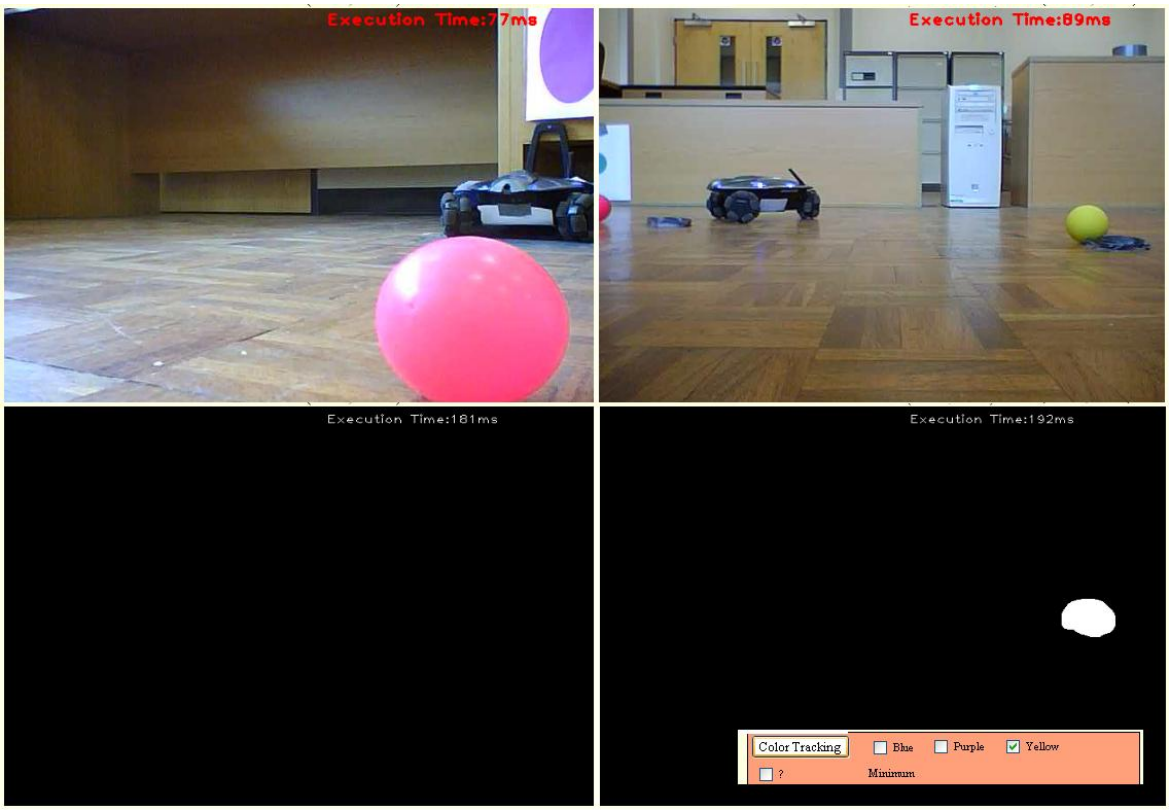

Figure 5-16 Yellow color tracking

The detected areas in the binary images are displayed in white. When the HSV min values are set in zero and the HSV max values in the max value (256) everything is detected (Figure 5-17).

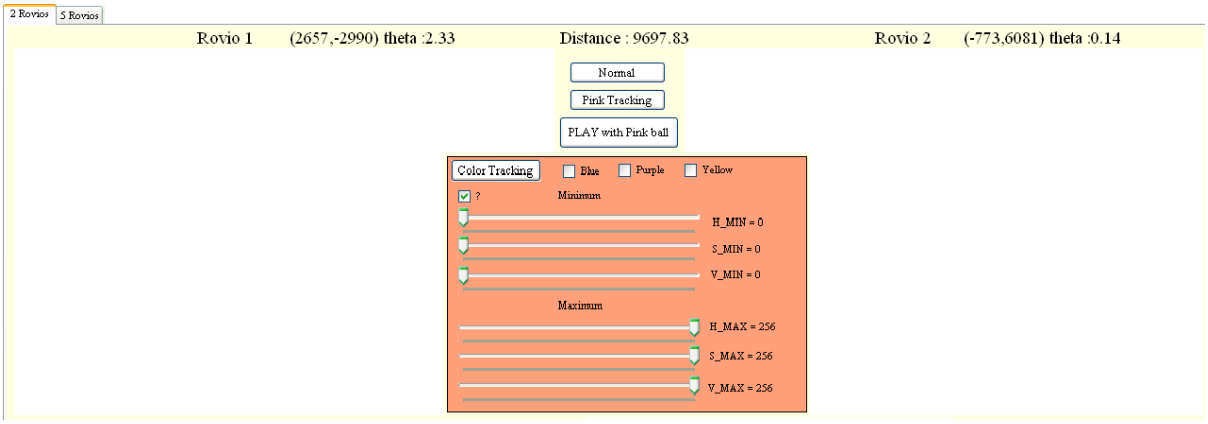

Figure 5-17 Detect everything

From these positions, by narrowing the range specific colors are detected and discarded. Figure 5-18 detects the colors between HSV_MIN= {36, 25, 64} and HSV_MAX= {225, 242, 222}.

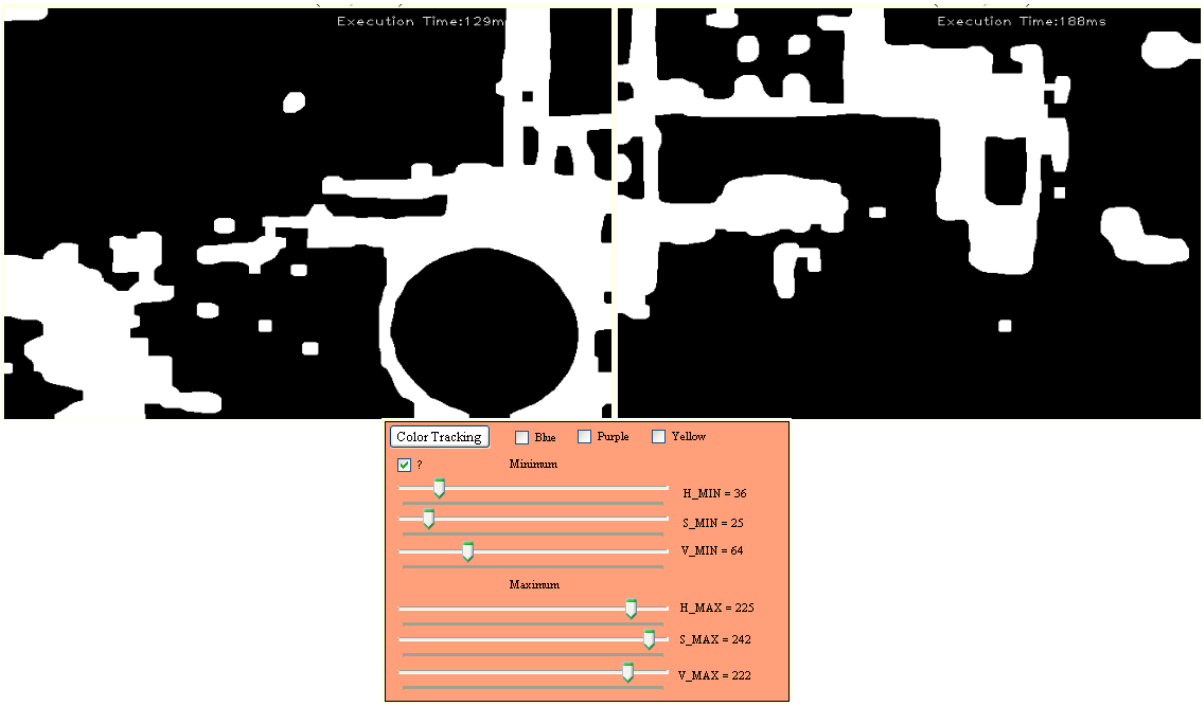

Figure 5-18 Narrowing the HSV range

Experimenting with the values of the trackbars that adjust the HSV values, we find out that one range that detects the current pink color is the HSV_MIN= {143, 43, 172} and HSV_MAX= {187, 183, 256} (Figure 5-19).

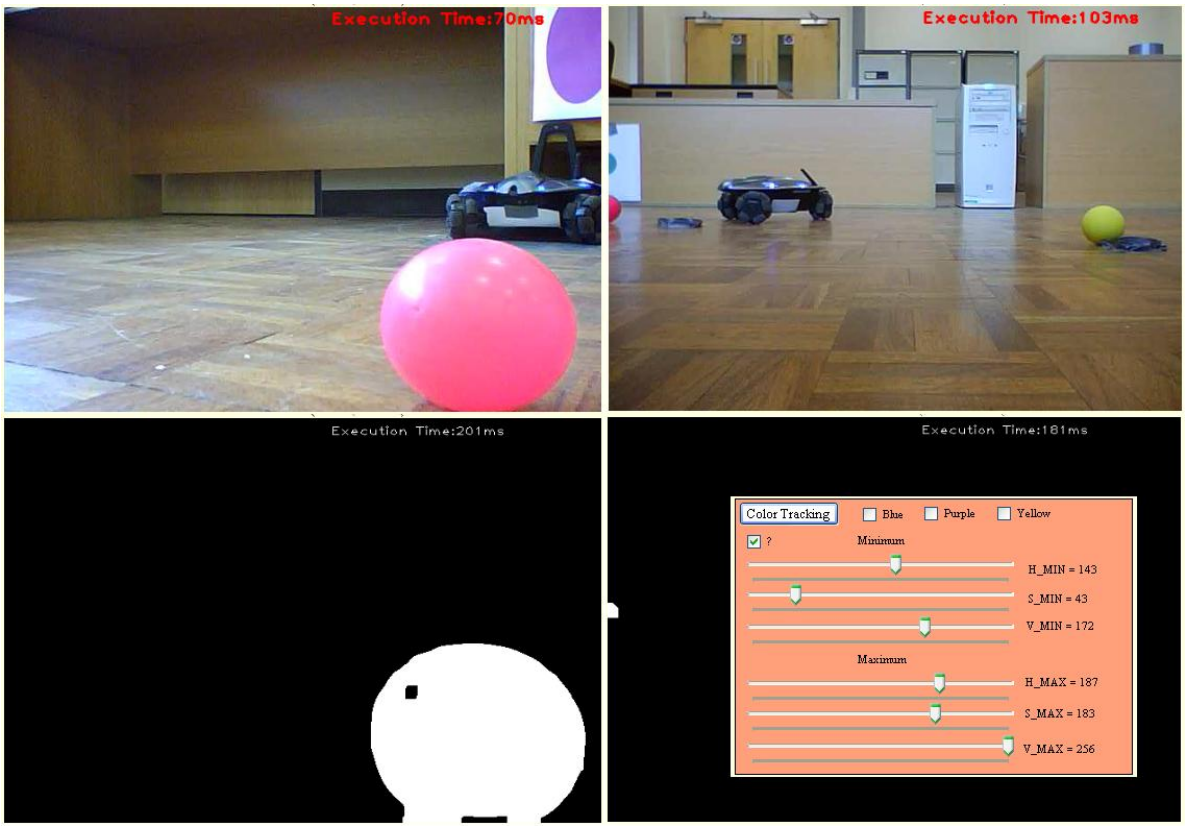

Figure 5-19 Adjusting the range

5.2.1.14 CIRCLE DETECTION AND DYNAMIC COLOR RANGE ADJUSTMENT PANEL

Figure 5-20 shows the circle detection and the dynamic color range adjustment panel. The "Circle Detection 1" button enables/disables the circle detection for current Rovio 1. The control next to this button displays the number of circles for detection (Figure 5-22). The box below displays the centres and the radiuses of 3 detected circles. All the important parameters for the circle detection algorithm are displayed on the right side of the panel. The threshold parameter is very important and can greatly change the result of the detection. After thresholding the original image smoothing is following using a Gaussian filter whose parameters can be adjusted from the four boxes below it. The canny parameters are also adjustable in the corresponding panel. After all this preparatory stage, the Circular Hough detection algorithm is applied whose parameters are adjustable from its panel on the extreme right of the panel.

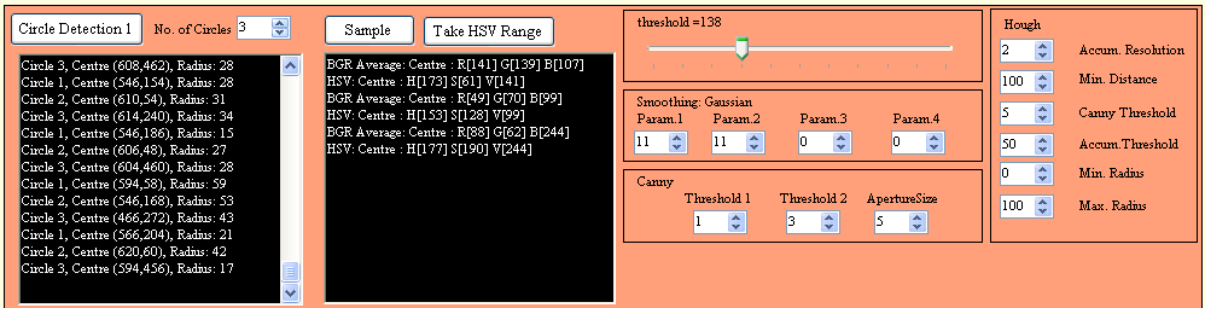

Figure 5-20 Circle Detection panel

By adjusting these parameters properly the circle detection functions effectively locking real circular shapes. When the centre and the radius are steady (notice that from the image and the first text box which displays the coordinates of the centre and the radius) the sampling is taking the color values of the detected area and displays them in the second text box. A BGR average is being calculated while its transformation to the HSV format is useful because based on that the new color range is being set.

Figure 5-21 shows which pixels are being used in the color averaging. The "Sample" button has to be pressed when the algorithm detect a circle steadily. This button enables the algorithm which calculates the average. The colors of nine pixels are taken for efficiency reasons. The transformation into HSV format happens after calculating the average of these 9 pixels in BGR format.

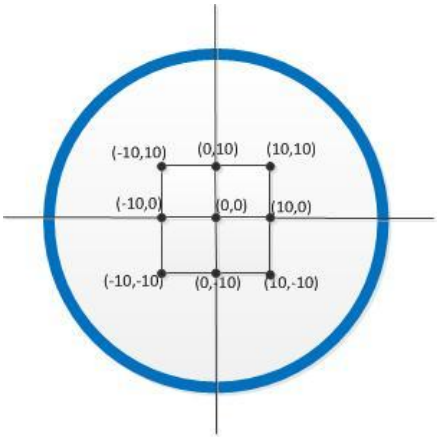

Figure 5-21 Pixels used in the color averaging

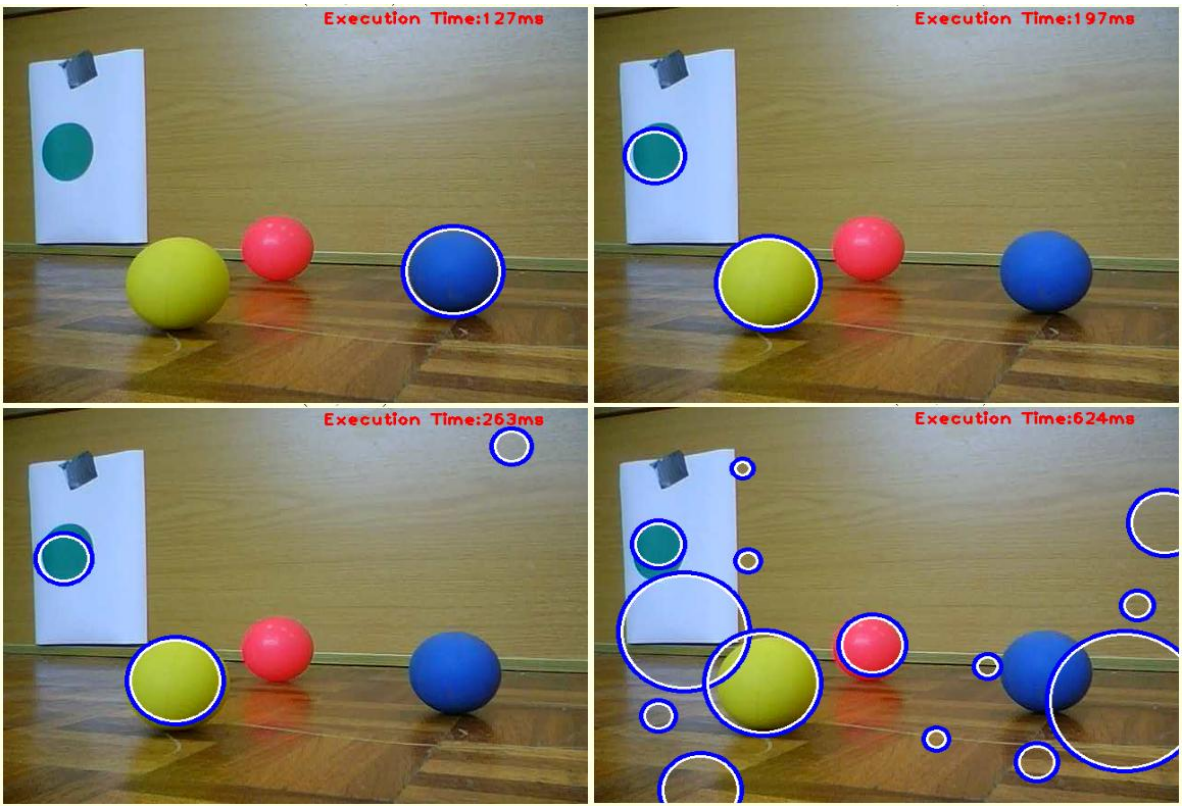

Figure 5-22 One, two, three and more circle detection

By setting the threshold value in 180, the light green ball is being locked (Figure 5-23). By pressing the “Sample” button the BGR average is: R [148] G [205] B [132] and its transformation into HSV is: H [137] S [90] V [205].

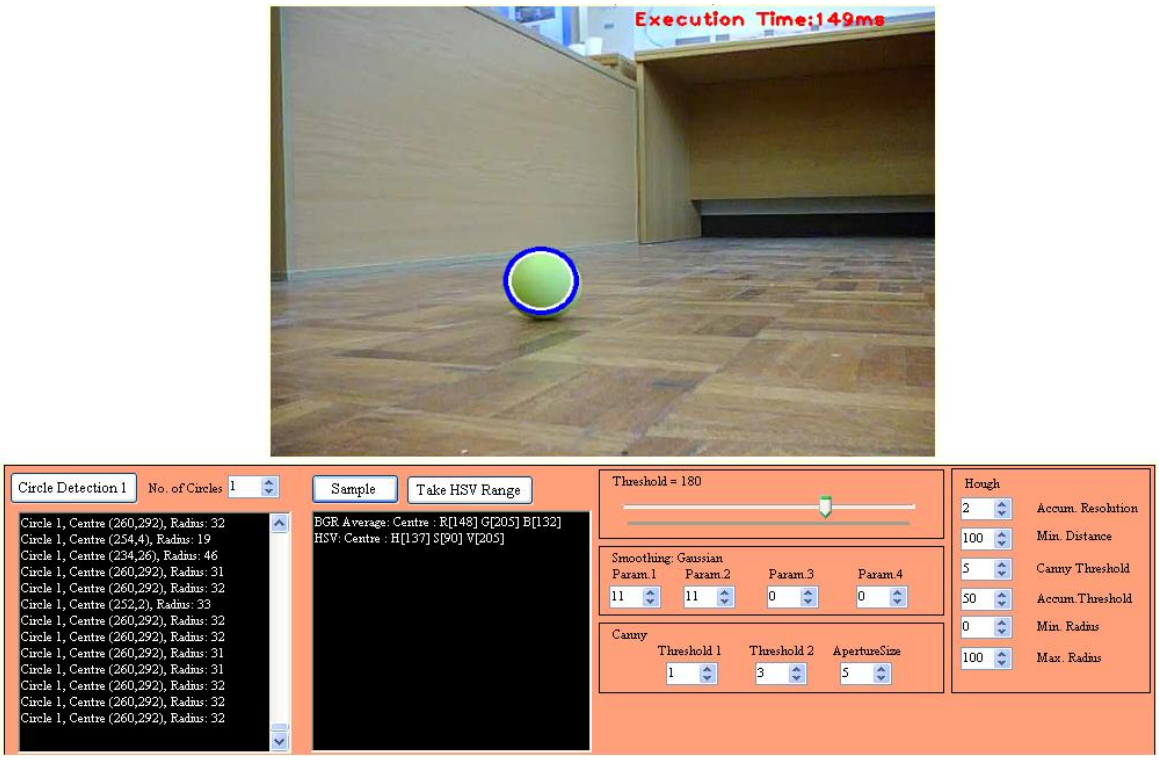

Figure 5-23 Ball detection

Figure 5-24 shows the importance of the right set up of the parameters. Only by changing the initial threshold and the smoothing parameters the algorithm is not able anymore to detect the ball in exactly the same image with the same light conditions (compare Figure 5-23 and Figure 5-24).

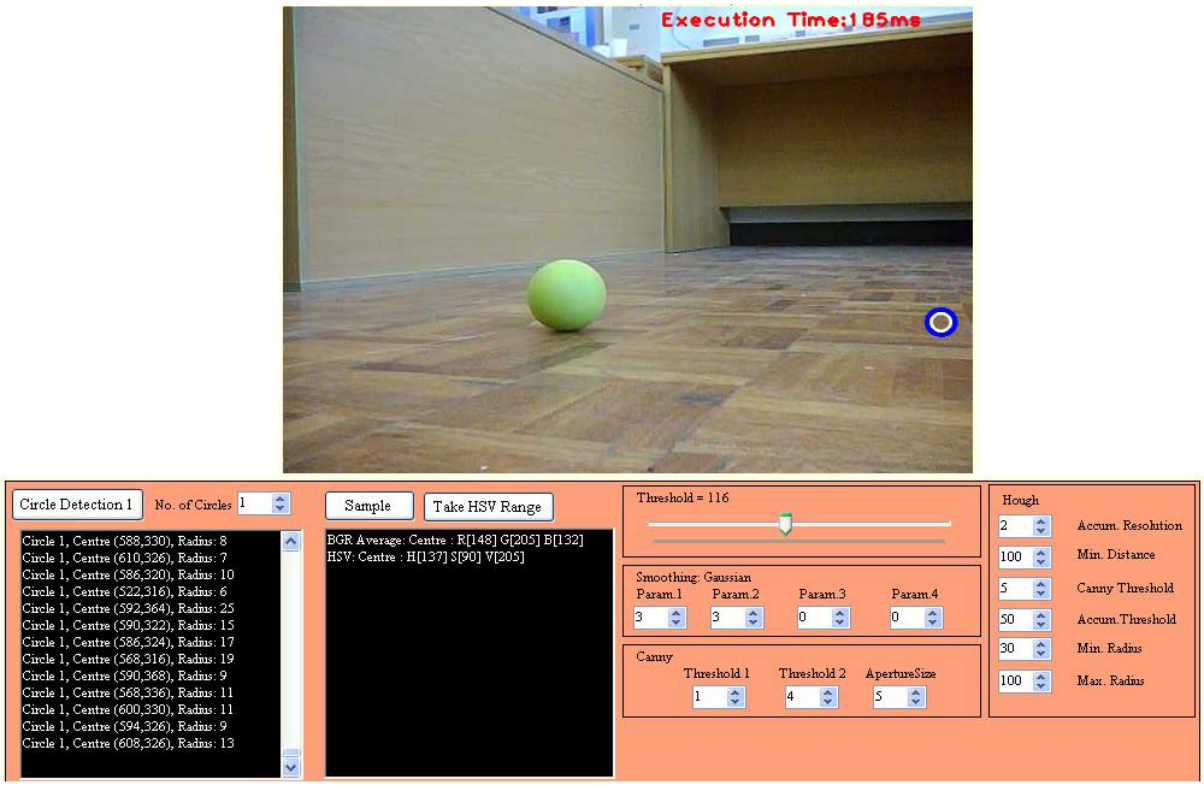

Figure 5-24 Importance of parameters in detecting the ball

By pressing the “Take HSV Range” a new HSV range in the Color Tracking panel is automatically being set up (Figure 5-25). If the result is not good and the binary image has a lot of noise it is preferable to manually correct the range. The effectiveness of the dynamic range set up method can be checked from the binary images by pressing the “Color Tracking” button (Figure 5-25). When the result is the desired one the “find new ball” button, from the first described panel (Figure 5-3), inputs the current HSV range in a new method which detects this color range and gives Rovio the appropriate commands to approach the new detected ball (Figure 5-26).

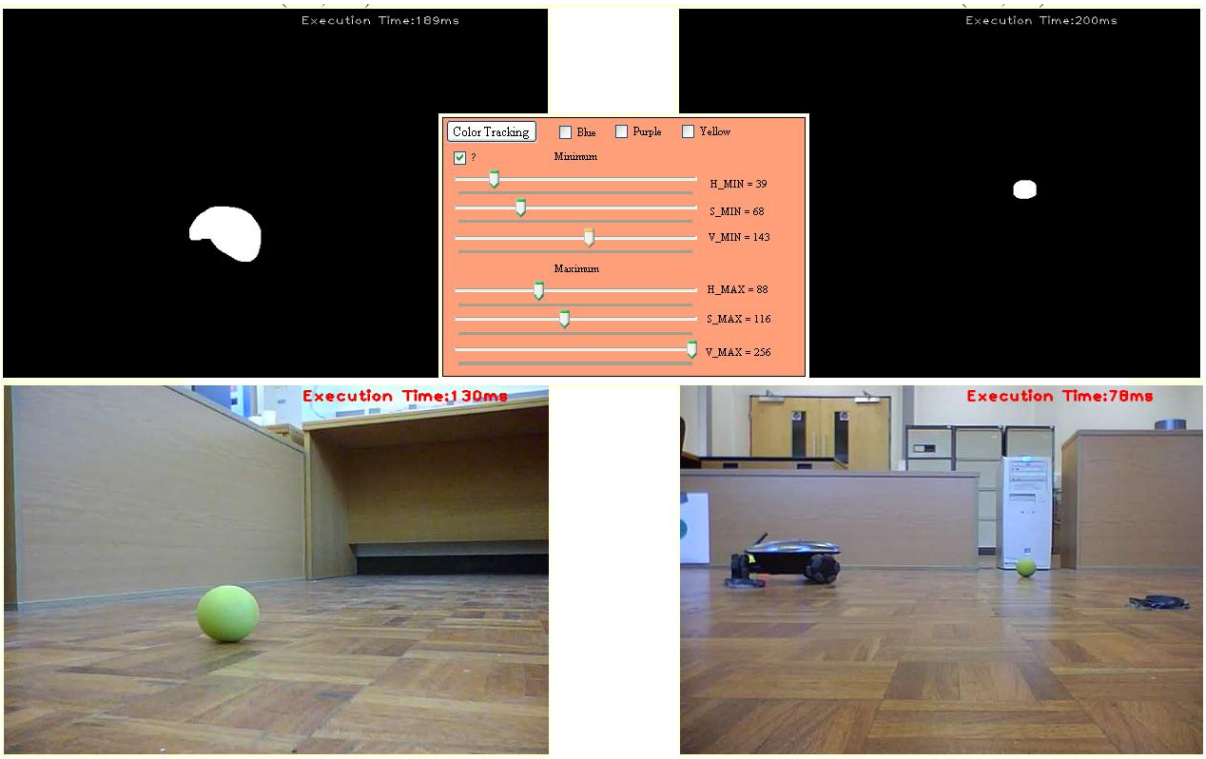

Figure 5-25 Adjust manually the range

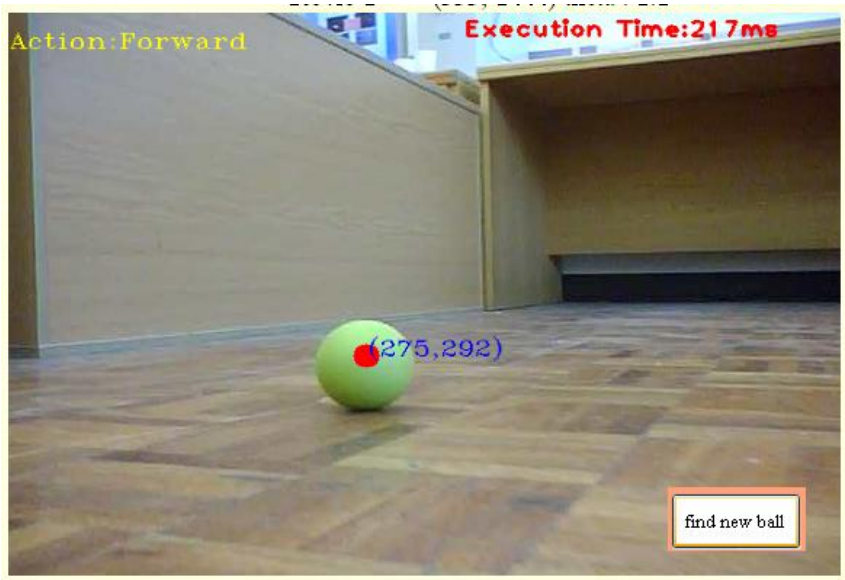

Figure 5-26 Find the new ball

5.2.2 TAB 2

Figure 5-27 shows Tab 2 which controls the five Rovios. The “Enable” button enables the threads that fetch the images from all the Rovios. This button was added because when Tab 2 is active the execution time of the image processing algorithms in Tab 1 raises significantly while the performance decreases.

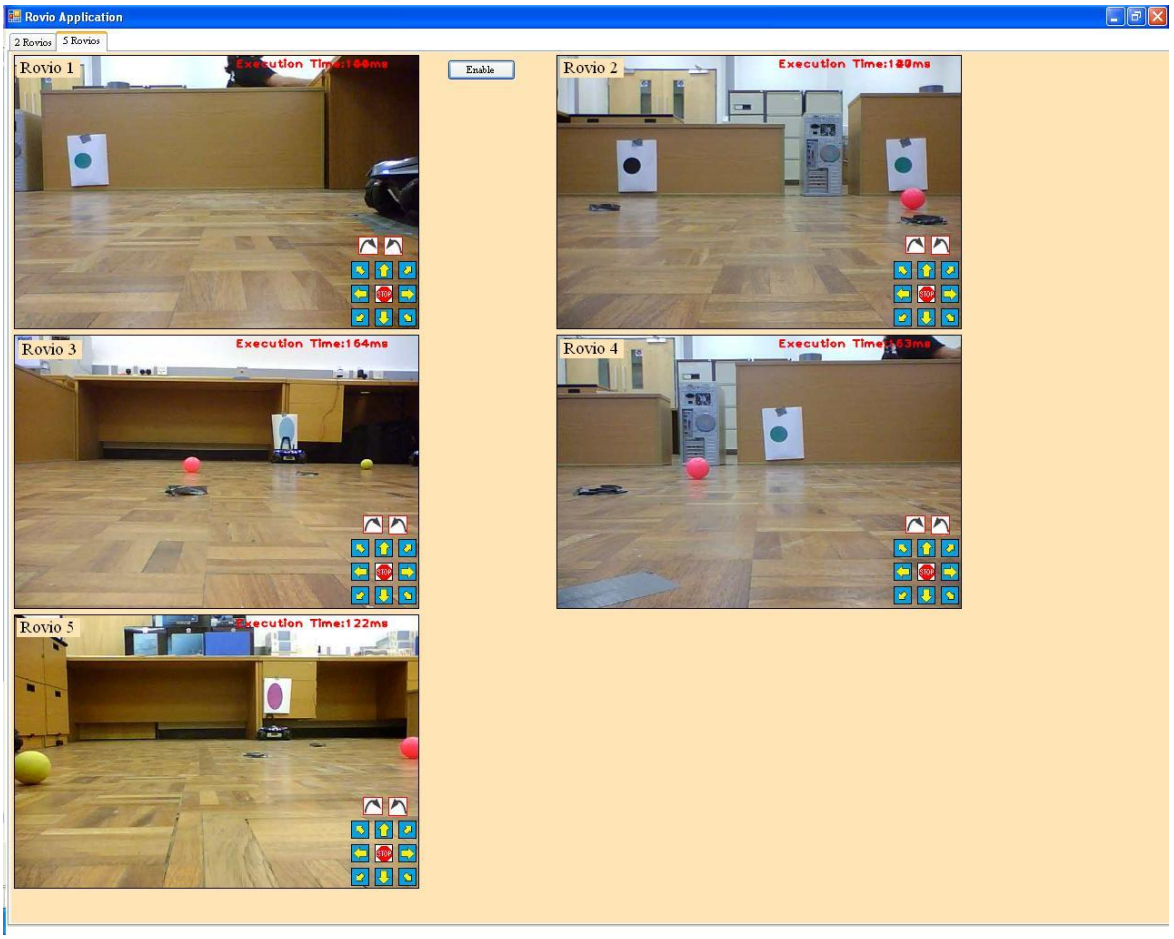

Figure 5-27 Tab 2

5.3 THREADING

The developed application has two types of threads; the threads for tab 1 (Figure 5-28) and the threads for tab 2 (Figure 5-29). By default the threads for tab 1 are enabled. To activate the threads for tab 2, the “Enable” button has to be pressed. As it was mentioned earlier, this button was added for efficiency reasons.

For tab 1 and tab 2, the buttons for manual control are being served via the main WinForm thread. In tab 1, the image acquisition and all the image processing methods are being served by the two threads that acquire the images from Rovio 1 and 2. The positioning thread is reading the coordinates of the current two Rovios and displays them and their between distance. The battery monitoring thread reads the level of battery and displays their level using the progress bars. The

wander methods for the two Rovios are being enabled/disabled by the corresponding buttons, also depicted in Figure 5-28.

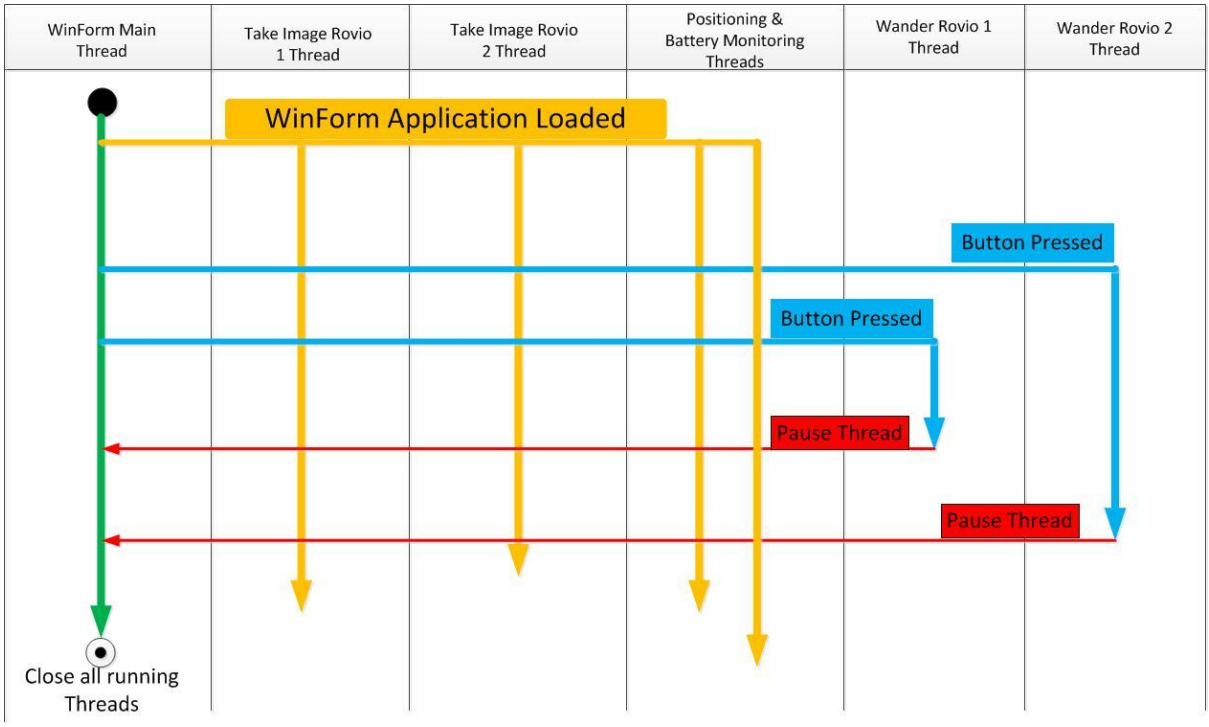

Figure 5-28 Threads in Tab 1

For tab 2, the threads that acquire the images for Rovio 1 and 2 continue to run, but now the images are being displayed in the two first image boxes in this tab, and three more threads that acquire the images for Rovios 3, 4 and 5 are being enabled (Figure 5-27 and Figure 5-29).

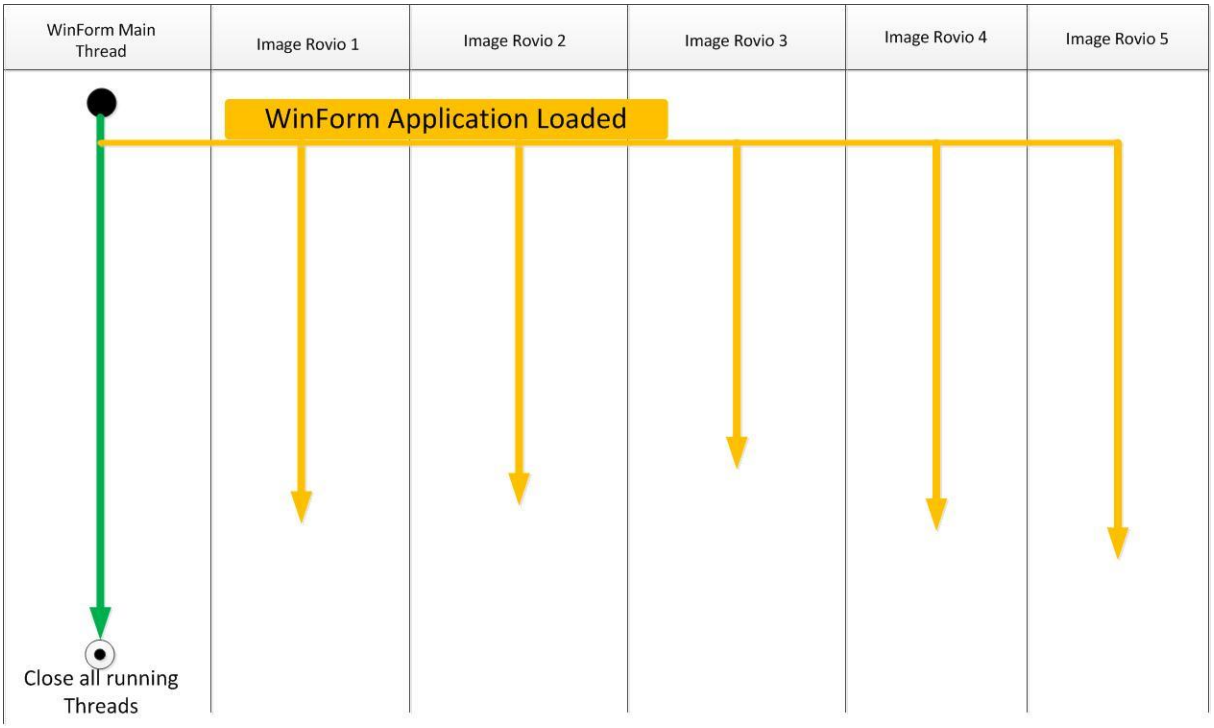

Figure 5-29 Threads in Tab 2

5.4 SUMMARY

This chapter showed in which way the methods have been incorporated in the GUI and how it controls and triggers different autonomous or semi-autonomous agent tasks.

The next chapter tests and evaluates the functionality of the GUI and the methods that are triggered from it.

6 TESTING & EVALUATION

In the beginning, the execution time and the number of frames per second will be calculated for the various image processing techniques that were developed (Normal, Edge Detection, Segmenting, Color Tracking and Circle Detection). After that, the main autonomous missions of the agents will be tested and evaluated. Evaluating the performance of such frameworks is notoriously difficult among other reasons because of the instability of the light conditions in the room that the experiments took place.

6.1 EXECUTION TIME

The measurement of the execution time was implemented with the use of the Stopwatch class. The stopwatch operator starts to count in the beginning of execution of the loop of the image acquisition thread and stops in the end of the each loop (Figure 5-8). Furthermore, the execution time is displayed on the upper right corner of each image. From this number the number of frames per second (FPS) for each image processing method can be approximated.

6.1.1 EXECUTION TIME IN TAB 1

In this chapter, the number of frames per second of each method will be calculated from the displayed execution time (in millisecond).

6.1.1.1 NORMAL EXECUTION

Table 6-1 has 10 values of the Execution Time (ET) in normal mode from current Rovio 1 and 2. The average ET is being calculated from these values, and finally the average FPS is being calculated from the ET average (1000ms/Average ET (ms) = Average FPS).

Table 6-1 Average ET and FPS in normal mode

Rovio 1	Rovio 2
98	111
118	79
94	77
92	63
77	72
88	59
93	77

147	78
98	65
74	65
Average ET: 97.9 ms	**Average ET: 74.6 ms**
Average FPS: 10.21	**Average FPS: 13.4**

6.1.1.2 EDGE DETECTION

Using the default values Thresh=70 and ThreshLinking=60.

Table 6-2 Average ET and FPS in default edge detection

Rovio 1	**Rovio 2**
90	93
92	76
101	102
117	117
97	142
115	115
90	147
92	137
108	85
93	124
Average ET: 99.5 ms	**Average ET: 113.8 ms**
Average FPS: 10.05	**Average FPS: 8.78**

6.1.1.3 PINK TRACKING

In the following experiment, both Rovios have the pink ball in their line of sight. The pink tracking and the segmenting methods have almost the same FPS that is the reason that measurements from the segmentation method are not included. This happens because the segmenting method comprises of the same image processing algorithm with the rectangles added in the end of the processing (Compare Figure 5-9 and Figure 5-12).

Table 6-3 Average ET and FPS while tracking pink

Rovio 1	**Rovio 2**

178	179
197	182
189	164
165	125
163	186
202	112
219	189
234	145
172	200
217	161
Average ET: 193.6 ms	**Average ET: 164.3 ms**
Average FPS: 5.16	**Average FPS: 6.09**

6.1.1.4 COLOR TRACKING

Color tracking in a random manual adjusted range (HSV_MIN={31,41,30}, HSV_MAX={216,207,228}).

Table 6-4 Average ET and FPS while tracking colors

Rovio 1	**Rovio 2**
199	155
118	97
134	138
168	133
202	130
134	104
202	146
142	144
142	178
185	180
Average ET: 162.6 ms	**Average ET: 140.5 ms**
Average FPS: 6.15	**Average FPS: 7.11**

6.1.1.5 CIRCLE DETECTION

Default parameters (Figure 6-1) and number of Circles to detect for Rovio 1.

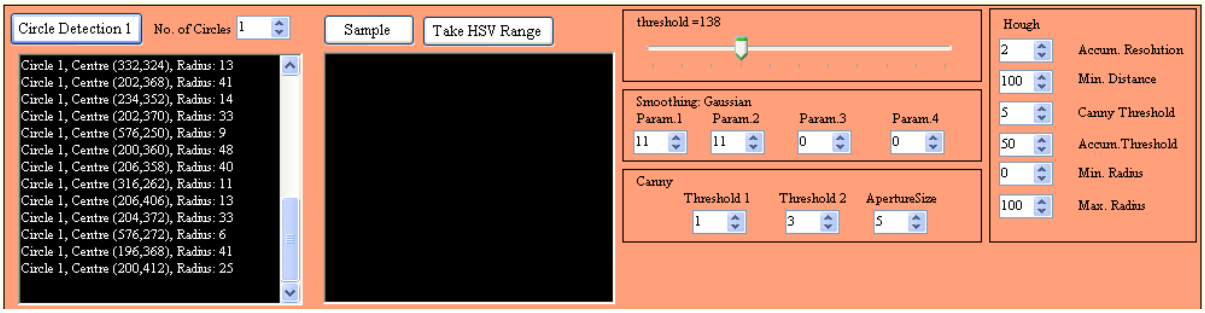

Figure 6-1 Default parameters for Circle Detection

Table 6-5 Average ET and FPS in default circle detection

Rovio 1 (No. Of Circles:1)	Rovio 1 (No. Of Circles:5)
171	260
271	274
207	273
261	206
200	214
161	304
284	299
174	317
191	282
273	299
Average ET: 219.3 ms	**Average ET: 272.8 ms**
Average FPS: 4.55	**Average FPS: 3.66**

Altered parameters (Figure 6-2) and number of circles to detect, again for Rovio 1.

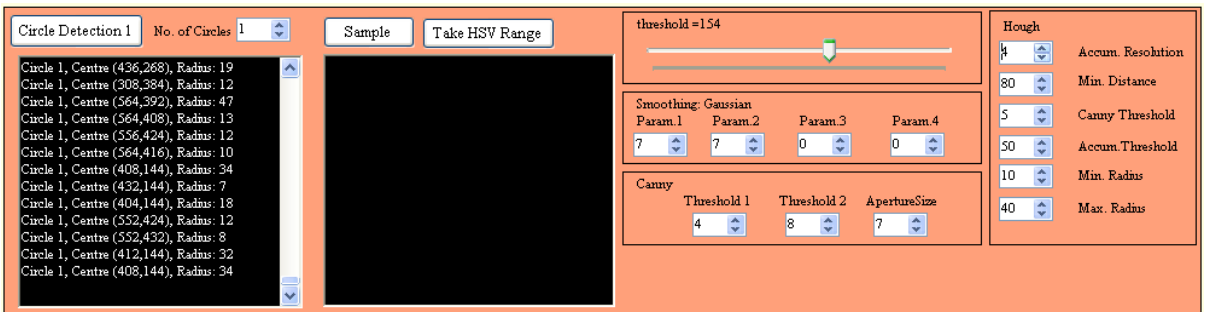

Figure 6-2 Altered parameters for Circle Detection

Table 6-6 Average ET and FPS in altered circle detection

Rovio 1 (No. Of Circles:1)	Rovio 1 (No. Of Circles:5)
227	301
218	322
242	264
261	301
212	290
283	278
228	320
243	311
239	274
240	299
Average ET: 239.3 ms	**Average ET: 296 ms**
Average FPS: 4.17	**Average FPS: 3.37**

6.1.2 Results

In the four first measurements the execution times for Rovio 1 and 2 for four different modes is measured (Normal, Edge Detection, Pink Tracking and general color tracking). The result that is depicted in Figure 6-3 was expected because the threads that acquire the images from the Rovios are being executed asynchronously. This means that the average ET and as a result the FPS has to be very close for current Rovio 1 and 2.

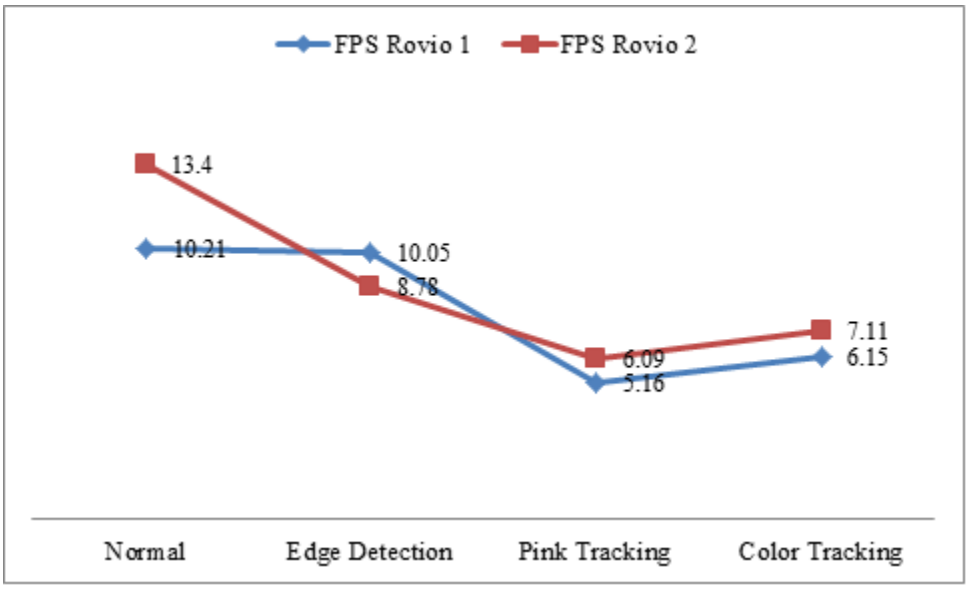

Figure 6-3 FPS comparison between Rovio 1 and 2

Figure 6-4 shows the average execution times for current Rovio 1 for different image processing methods of this project. As it was expected the quickest method is the simple display of the acquired image from the camera, unprocessed. Edge detection is slightly slower. Pink and Color tracking are very close. Circle's detection ET depends heavily in its parameters and especially in the number of the detected circles. The same results are depicted in Figure 6-5 (Normal operation is the quickest while Altered Circle detection (No. of Circles:5) is the slowest).

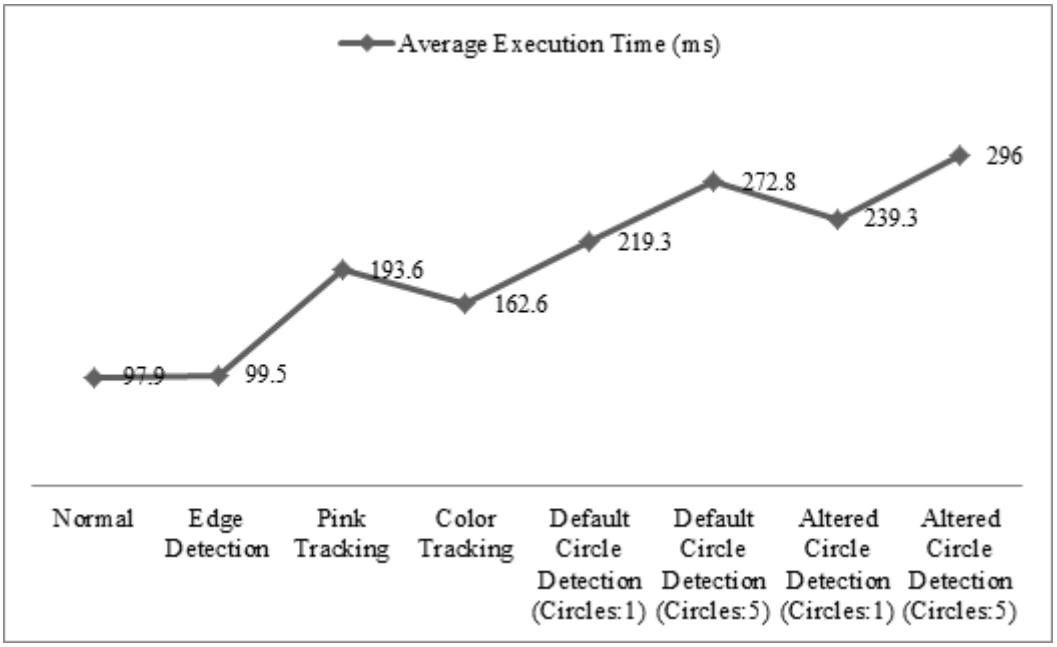

Figure 6-4 Average Execution Time (Rovio 1)

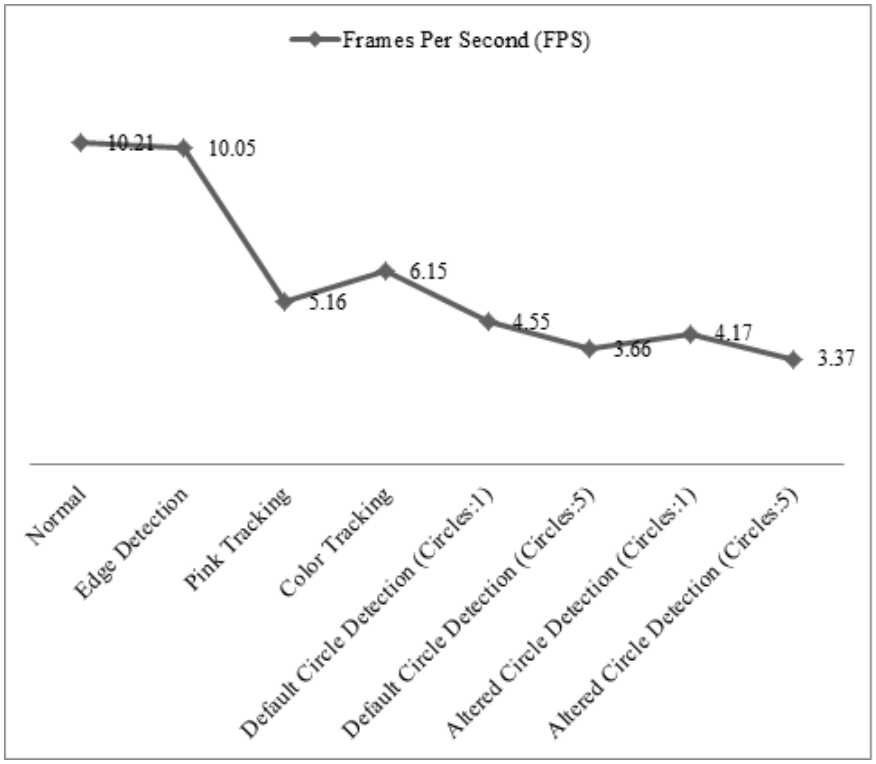

Figure 6-5 Average Frames Per Second (Rovio 1)

6.2 IR BASED WANDERING

Figure 6-6 shows Rovio while it is wandering around the room with the coloured balls and the walls as physical obstacles. In Figure 6-6 (a) the agent is docked. It starts moving straight (b), it reaches the wall (c) and turns left (d), continues straight and passes by the yellow ball (e), finds another docked agent (f) and turns again left towards the yellow ball (g and h), turns again left and moves straight (i).

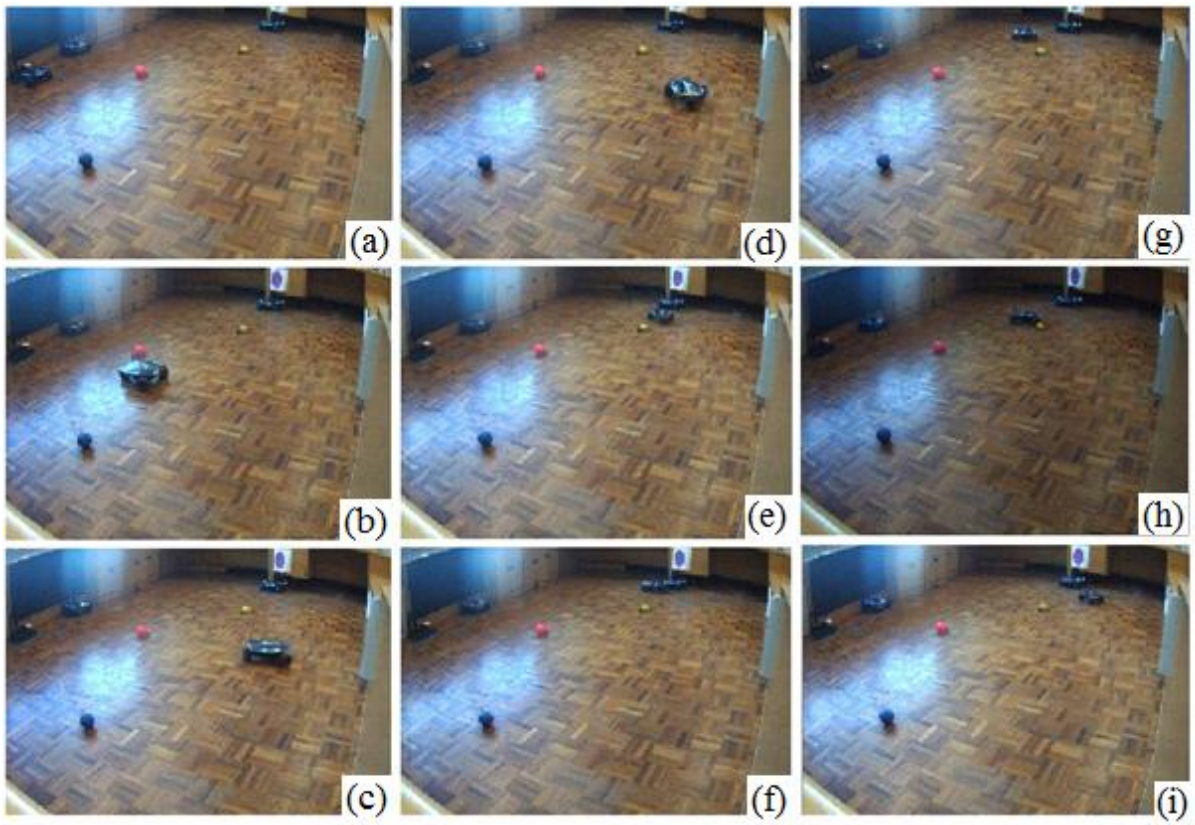

Figure 6-6 IR based wandering

With the current simple algorithm (described in 4.1.1) Rovio can autonomously wander in an unfamiliar environment while the receiving video provides useful information to the user. In case of collision, the user can effectively help the agent to overcome it using the manual controls.

6.3 FIND PINK AND YELLOW BALL

This chapter measures how much time Rovio wants to find and approach the ball. Also, the success rate of this mission in full operation speed is being calculated. In these experiments the pink ball is inside the sight of the current Rovio 1 (here Rovio 5) (Figure 6-7).

For these measurements Rovio 5 was used (Table 6-7). The pink ball is placed in a standard position in a distance from the agent (approximately 3 meters away). The time from the starting position till Rovio reaches the ball is being measured.

Figure 6-7 View from docked Rovio 5 (find pink ball)

Table 6-7 Rovio 5 find pink ball measurements (line of sight)

No. of Experiment	No. of Rotations	Time	Result of Experiment
1	0	6.31 sec	Success
2	0	6 sec	Success
3	0	6.3 sec	Success
4	0	6.18 sec	Success
5	0	6.1 sec	Success

Figure 6-8 shows the actions of the Rovio in this experiment. The images on the left are the images that Rovio acquires and processes in order to complete its task. The images on the right are taken at the same time with the images on the left from an external camera. These figures shows how Rovio moves from its home position towards the ball.

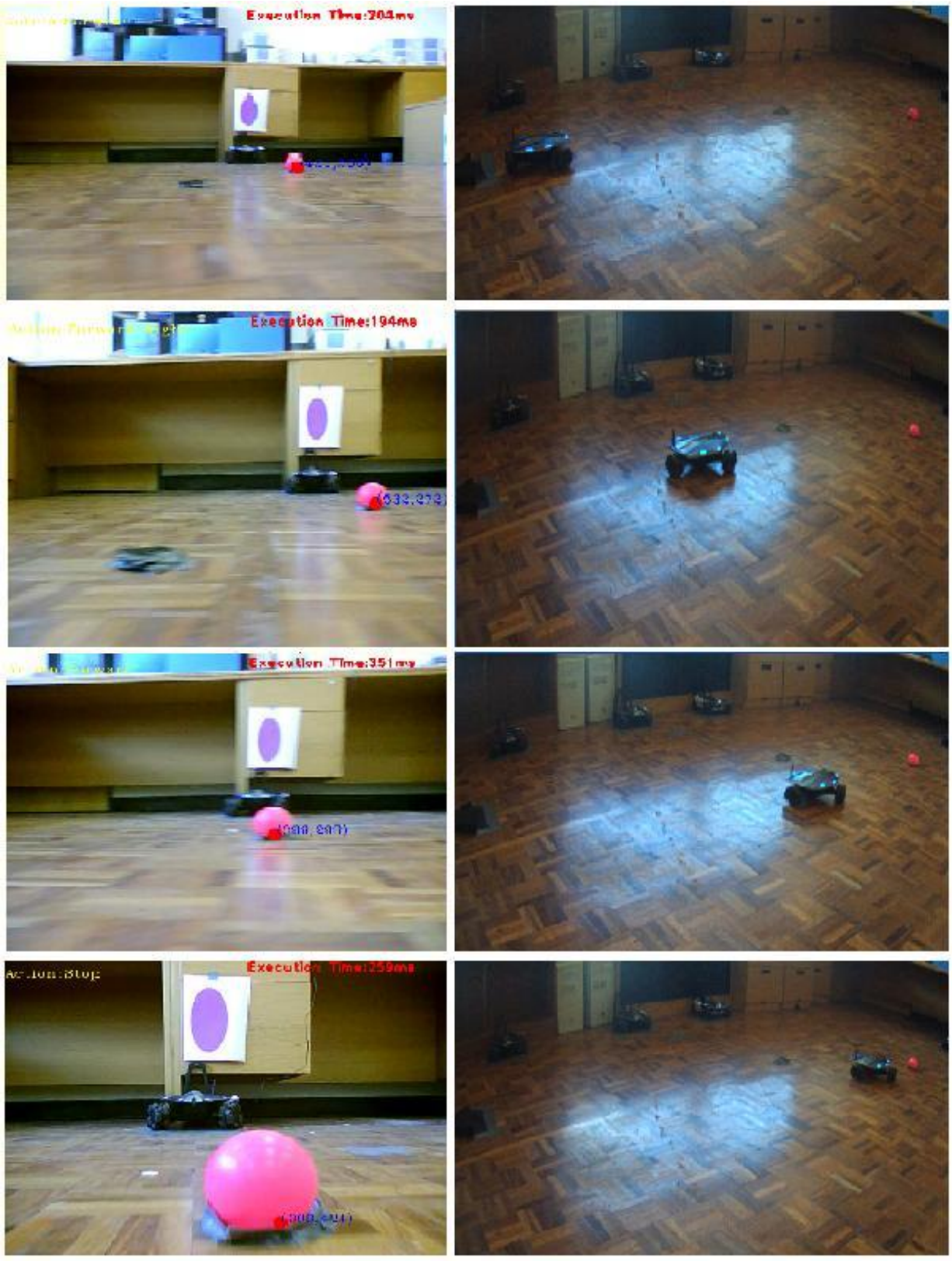

Figure 6-8 Find Pink Ball (Rovio 5 – Line of sight)

Now the same experiment is repeated without having the ball in the line of sight of the current Rovio 1 (here Rovio 1) (Figure 6-9). The agent has to rotate in order to "lock" the target in his view and then approach it.

Figure 6-9 View from docked Rovio 1 (find pink ball)

In comparison with the previous experiment a significant increase is noted in time mainly because of the rotation that is required in order to "lock" the target. The rate of success is again 100%. The no. of rotations and the time depend heavily on the light conditions of the room. The success rate may decrease as the light conditions change. For example, by turning off the lights in the room, it may be impossible for the agent to "lock" its target. In this occasion, helping manually the agent is one choice either by re-setting the HSV range that is looking for or by following the dynamic color range adjustment procedure as it is described in 5.2.1.14.

Table 6-8 Rovio 1 find pink ball measurements (non-line of sight)

No. of Experiment	No. of Rotations	Time	Result of Experiment
1	1	21.64 sec	Success
2	1	19.18 sec	Success
3	1	16.5 sec	Success
4	1	17.77 sec	Success
5	1	15.68 sec	Success

Figure 6-10 shows how Rovio searches the ball, by rotating, and then moves towards it.

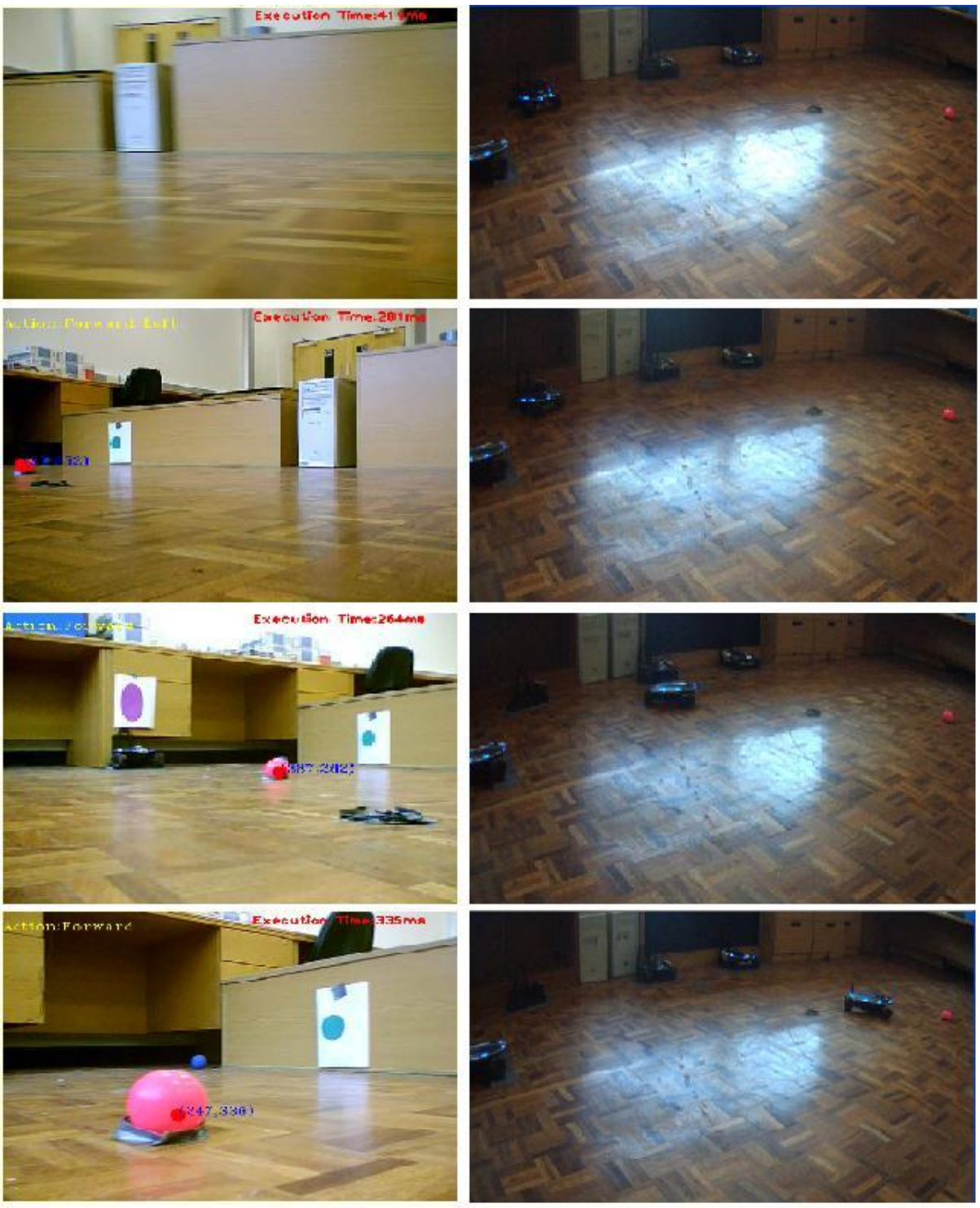

Figure 6-10 Find pink ball (Rovio 1 – Non-line of sight)

The same experiment with yellow ball follows.

Figure 6-11 View from docked Rovio 1 (find yellow ball)

The success rate is again 100%. However, it is noted that sometimes Rovio wants one full rotation to "lock" its target and sometimes just a quarter. This is the reason why time varies significantly.

Table 6-9 Rovio 1 find yellow ball measurements (non-line of sight)

No. of Experiment	No. of Rotations	Time	Result of Experiment
1	¼ Rotation	7.4 sec	Success
2	¼ Rotation	8.33 sec	Success
3	1	16.79 sec	Success
4	1	20.47 sec	Success
5	¼ Rotation	7.43 sec	Success

Figure 6-12 shows how Rovio searches the ball, by rotating, and then moves towards it.

Figure 6-12 Find Yellow ball (Rovio 1 – Non-line of sight)

6.4 CHANGE AGENT THROUGH MISSION

Here Rovio 1 will be assigned to find the pink ball. Before it completes its task, Rovio 4 will be assigned to complete its task by using the dynamic characteristic of the GUI to select and change current Rovios 1 and 2. In this example, current Rovio 1 is being swapped, Rovio 1 ("http://192.168.2.11") swaps with Rovio 4 ("http://192.168.2.15").

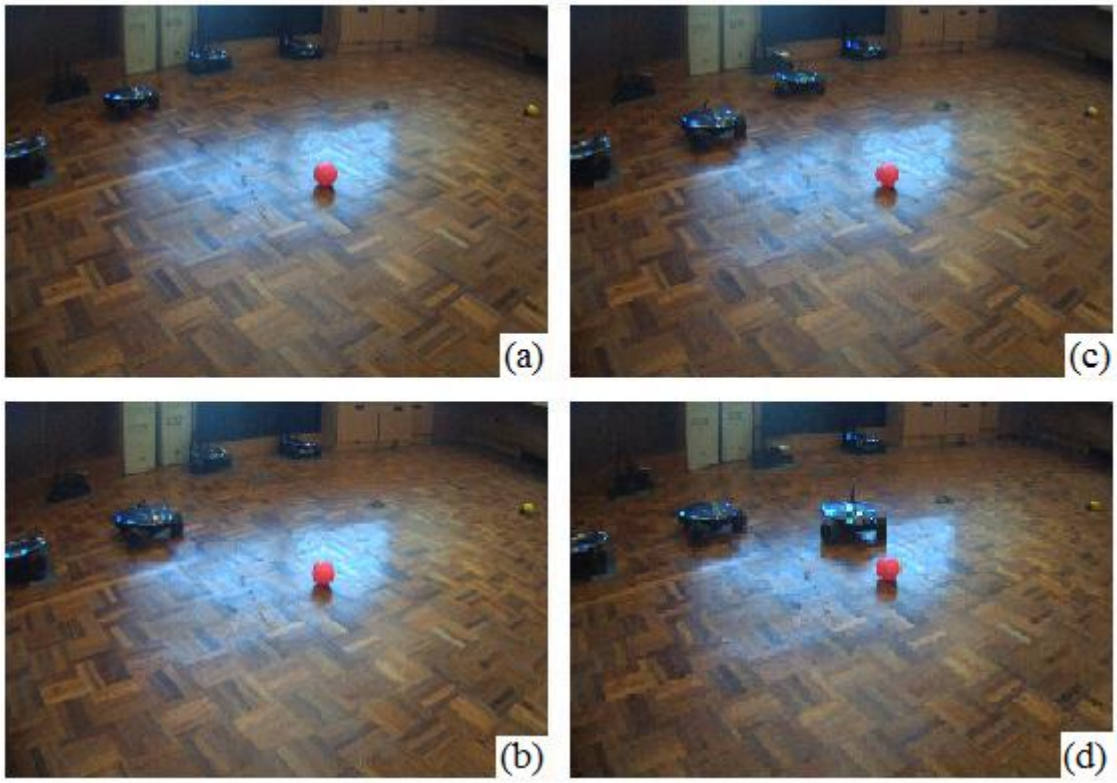

Figure 6-13 Find pink ball, swapping agents

As Figure 6-13 shows, Rovio 1 detects and moves towards the ball (a and b), at that moment Select Rovios panel (Figure 5-10) is being used in order to assign Rovio 4 as the current Rovio 1. As a result, Rovio 1 stops and Rovio 4 continues and fulfils its mission (find the pink ball) (c and d).

This was an example of this dynamic characteristic. Swapping agents can be done during any kind of mission that is implemented in this project. This semi-autonomous behaviour can help in difficult situation where one or more agents are not able to complete their task. The reasons for not completing their mission can vary from simple battery depletion issues till heavy collisions in isolated areas without visual contact from the operator. Therefore, having some agents standby can be useful and helpful.

6.5 Play with Pink Ball

This method described in 4.4. Two Rovios are involved to fulfil this scenario. Figure 6-14 shows these two agents in action. Current Rovio 1 moves towards the ball (a), when he finds the ball (b) it calls current Rovio 2 to find the ball (c). Rovio 2 finds the ball (d) and Rovio 1 searches again (rotates searching) the ball (e). Rovio 1 finds and kicks the ball (f). The ball moves away and Rovio 1 follows it (g) because the mass centre of the ball must be below a certain y coordinate level in

order to give permission again to Rovio 2. When this condition is fulfilled Rovio 2 takes again permission to move (g and h). Finally, in (i) Rovio 2 finds again the pink ball.

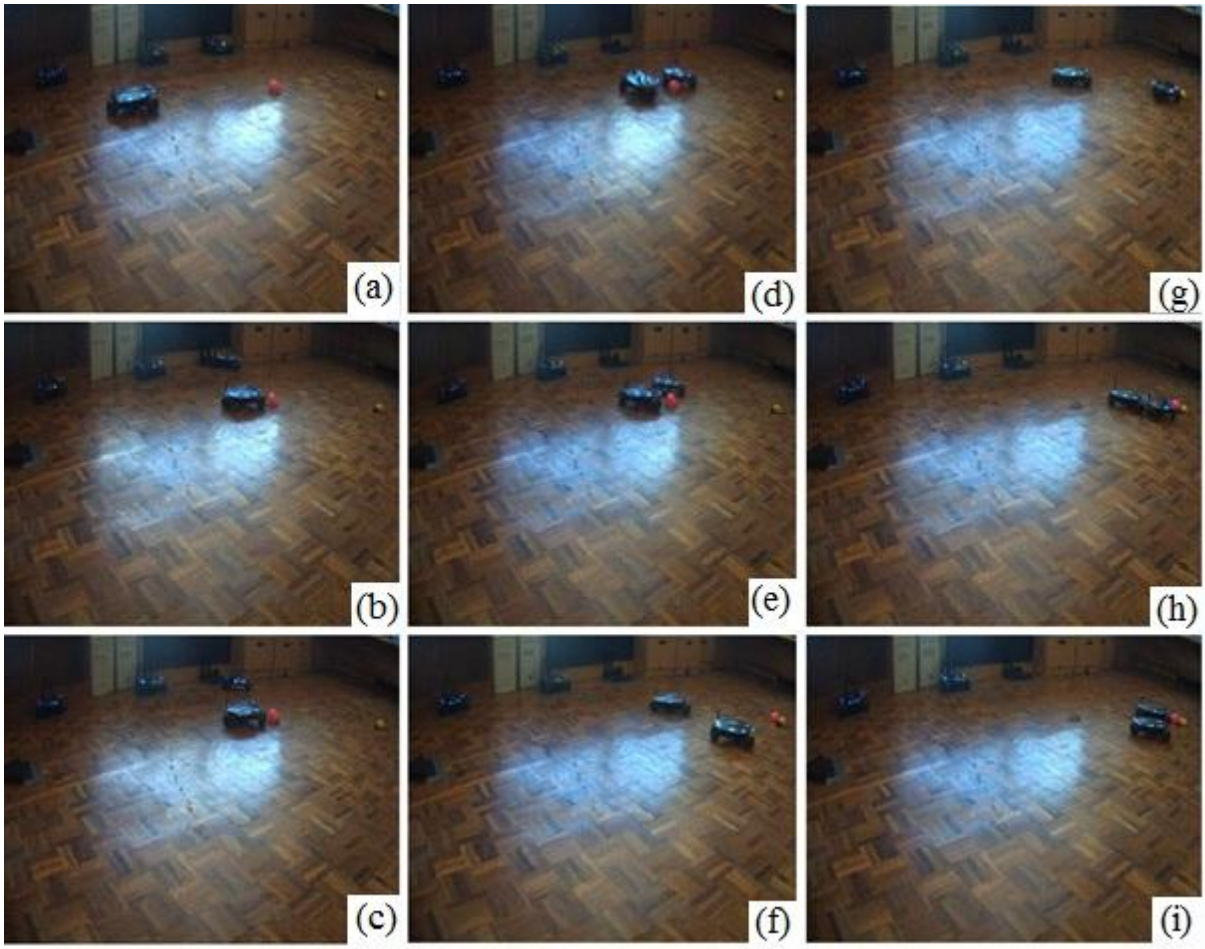

Figure 6-14 Play with the pink ball

6.6 Visual Servoing and Navigation Scenario

Visual servoing and navigation scenario described in 4.5. An agent finds the pink, the yellow, again the pink ball and drives home. Figure 6-15 and Figure 6-16 depict the actions of the agent. Again the images on the left are taken from the agent while the images on the right from an external camera which inspects the progress of the mission.

Rovio searches, finds and moves towards the pink ball. While he searches for the yellow ball, he touches (due to his rotation) and changes the position of the pink ball. He finds the yellow ball and moves towards it (last image of Figure 6-15).

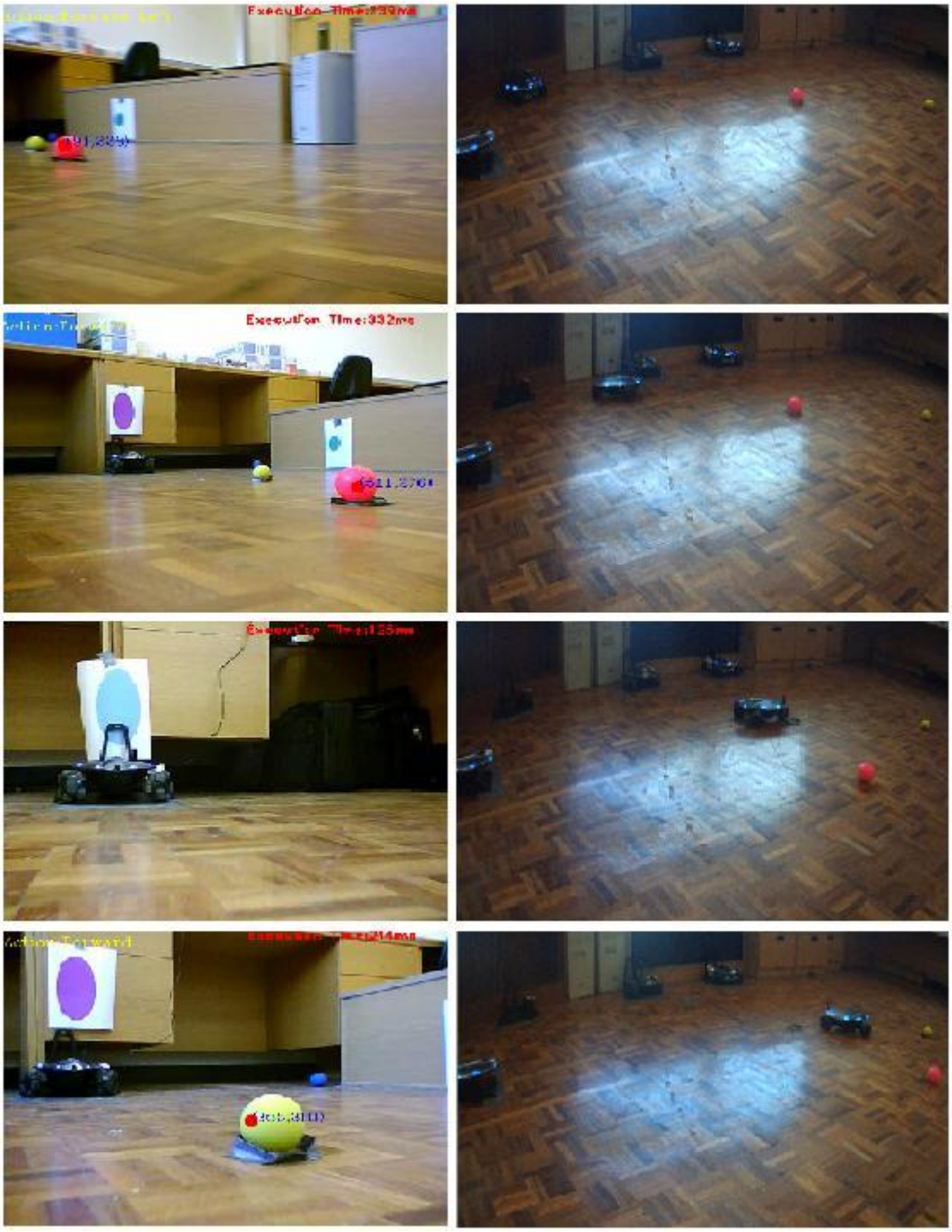

Figure 6-15 Mission 1 (part 1)

After finding the yellow ball, he searches the pink ball again (first image of Figure 6-16). He approaches the pink ball for the second time and after that he is heading for his base position where he finally docks (last image of Figure 6-16).

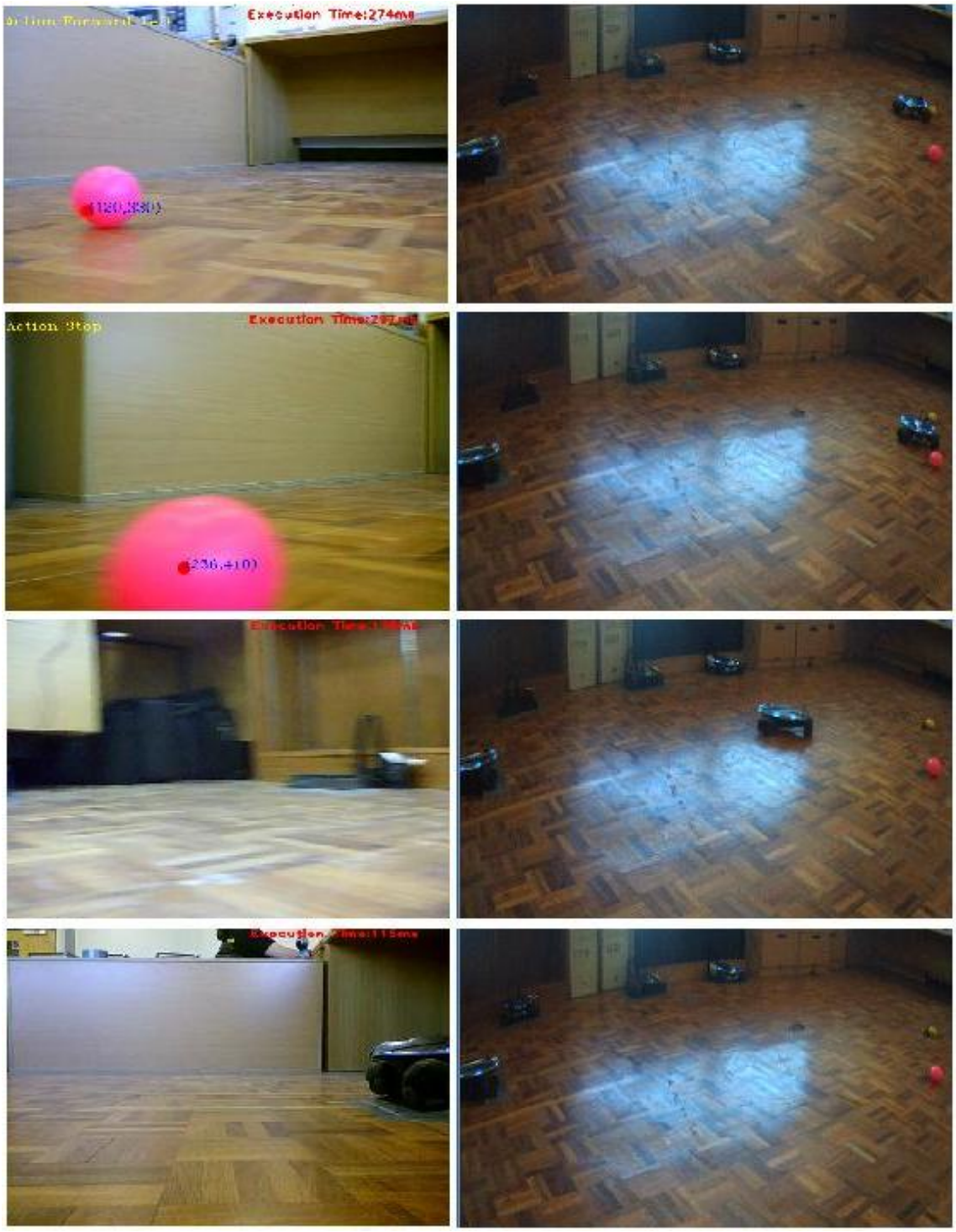

Figure 6-16 Mission 1 (part 2)

6.7 FIND NEW BALL

This chapter tests and evaluates the circle detection and the dynamic color range adjustment method in three different light conditions. The target object will be a blue ball (Figure 6-17).

6.7.1 EXPERIMENTS

6.7.1.1 EXPERIMENT 1

Experiment 1 will be made with all the light on in the room. From a distance of approximately 1 meter the agent is able to "lock" the target by changing the main Threshold value to 39. The first text box is displaying the Centre and the Radius of the detected circular shape (Figure 6-18).

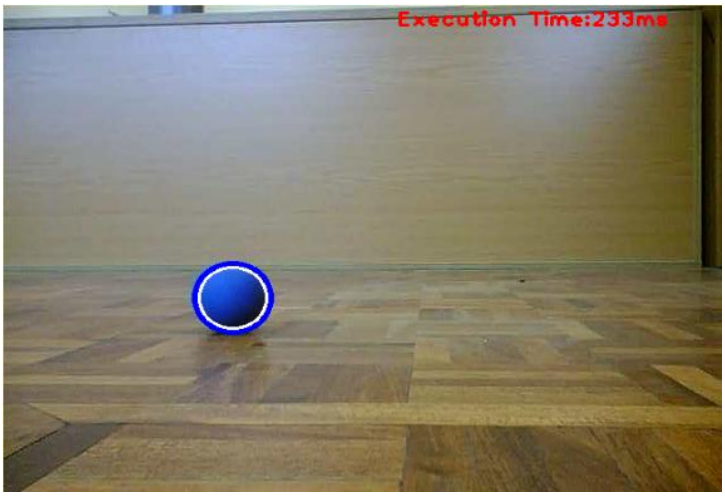

Figure 6-17 Target locked

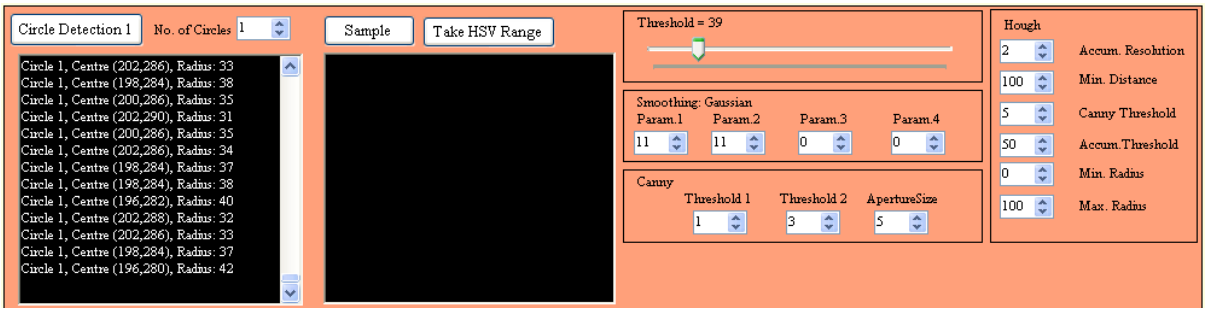

Figure 6-18 Threshold adjusted and Centre and Radius Displayed

When the target is "locked", the sampling procedure takes the color of the pixels of the central area of the detected area and calculates the average at it is described in 5.2.1.14 (Figure 6-19).

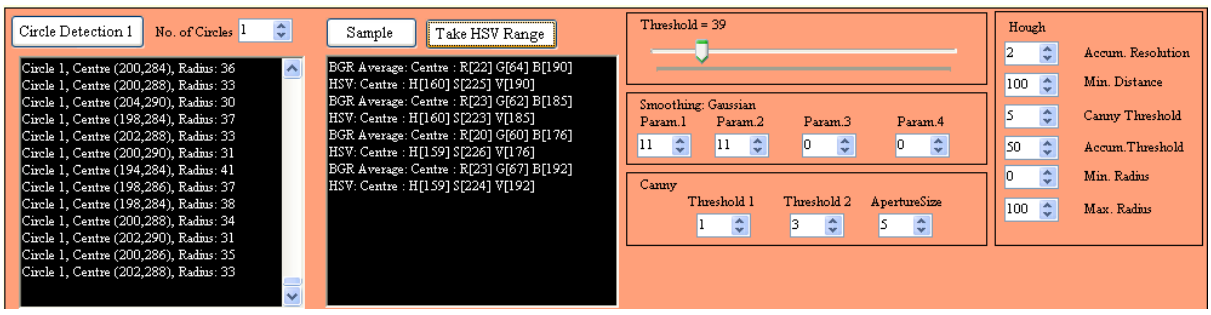

Figure 6-19 Sampling the color

After pressing the button "Take HSV Range", the HSV bars in the Color tracking panel are automatically set. The results can be checked on the binary images by ticking the "?" checkbox

(Figure 6-20). A simple comparison of Figure 6-20 and Figure 6-17 demonstrates that the blue ball will be detected successfully.

Figure 6-20 Checking the result of the automated color range adjustment

The panel on the right of Figure 6-20 shows the min and max HSV value that the “find new ball” method will use as input. Pressing the “find new ball” button Rovio will detect the blue ball and the auto-commanding method will lead the agent in front of the ball.

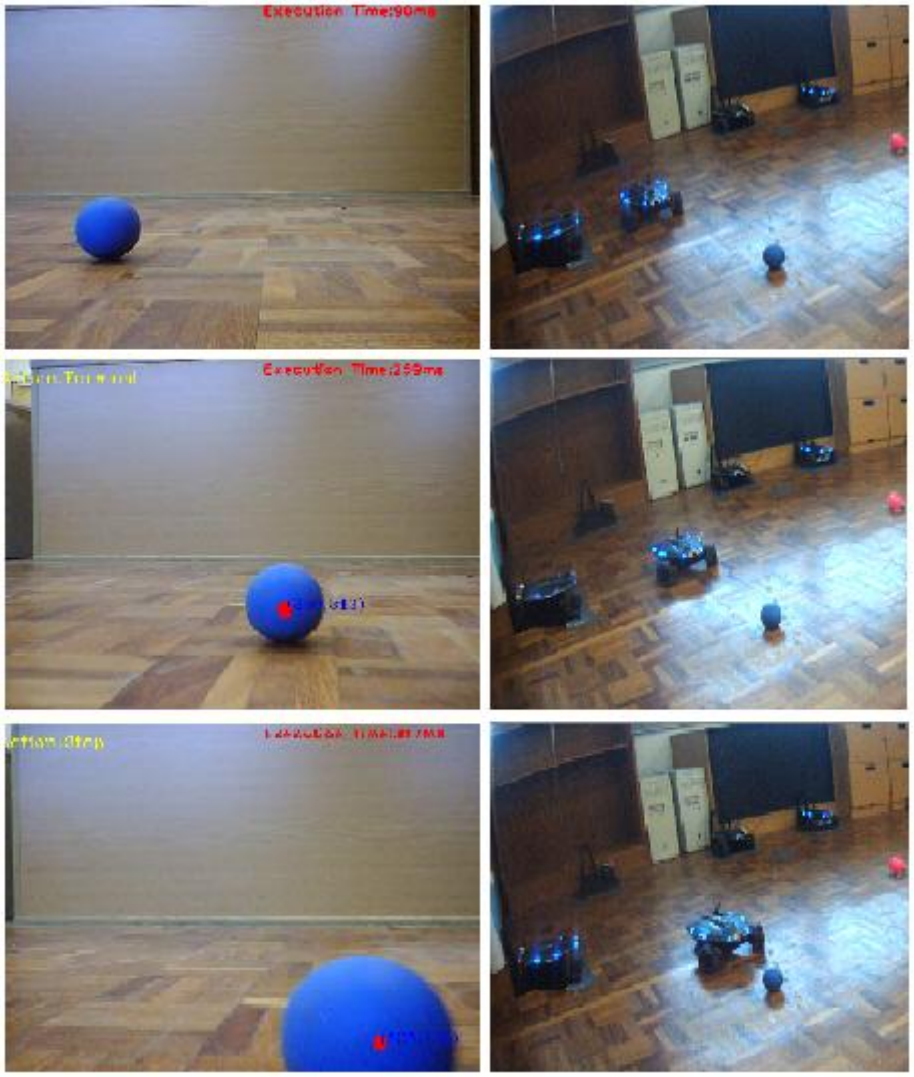

Figure 6-21 Find new ball

6.7.1.2 EXPERIMENT 2

By turning off some of the lights of the room, the HSV range changes as it adapts to the new light conditions (color tracking panel in Figure 6-22). From the detected area of the ball can be sensed that the "find new ball" method will be again successful (Figure 6-22).

Figure 6-22 Checking the result of the automated color range adjustment after the light condition changing

6.7.1.3 EXPERIMENT 3

By turning off all the lights of the room, the HSV range changes again accordingly. Under these light conditions it is noted that the circle detection algorithm is not "locking" the target as steadily as it was used to in the previous light conditions. Moving the Threshold value to 21, the sampling color values will be the desired because the method is being executed efficiently. Again, from the detected area of the ball is sensed that the "find new ball" method will be executed with success.

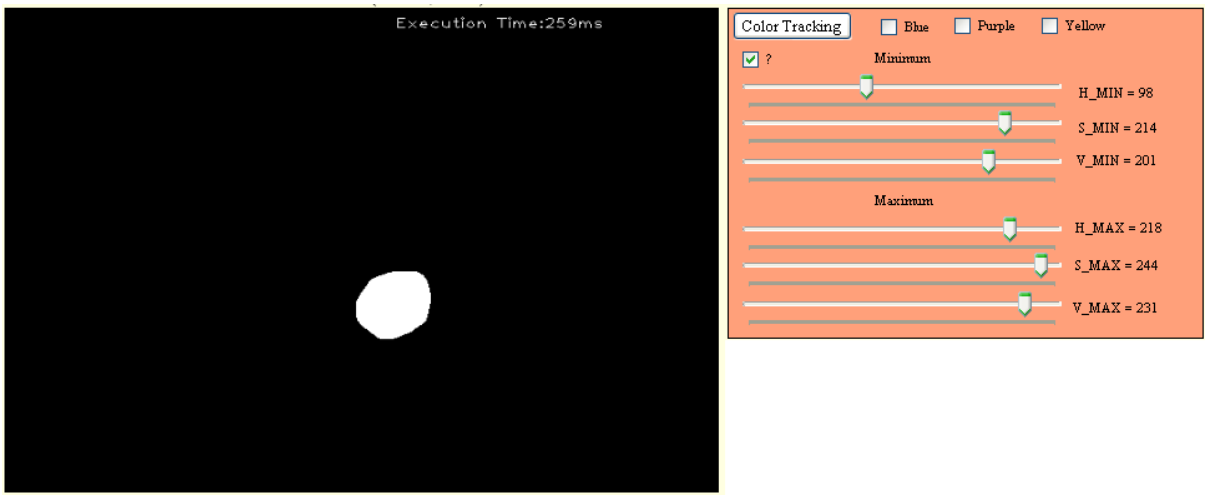

Figure 6-23 Checking the result of the automated color range adjustment after turning off all the lights in the room

The figure below illustrates the different light conditions that were used in the experiments from an external camera in the room and from Rovio.

Figure 6-24 Different light conditions for the experiment (top: Experiment 1, middle: Experiment 2, bottom: Experiment 3)

6.7.2 RESULTS

Table 6-10 presents the HSV min and max values that are automatically generated from this method. It is easy noticeable that the higher alteration is noticed in the Value parameter. This was expected because the parameter of Value describes the lightness or the brightness which was changed by closing the light in the room.

Table 6-10 HSV Color Ranges in the three experiments

	H_Min	H_Max	S_Min	S_Max	V_Min	V_Max
Experiment 1	100	220	213	244	176	206
Experiment 2	98	218	212	243	223	253
Experiment 3	98	218	214	244	201	231

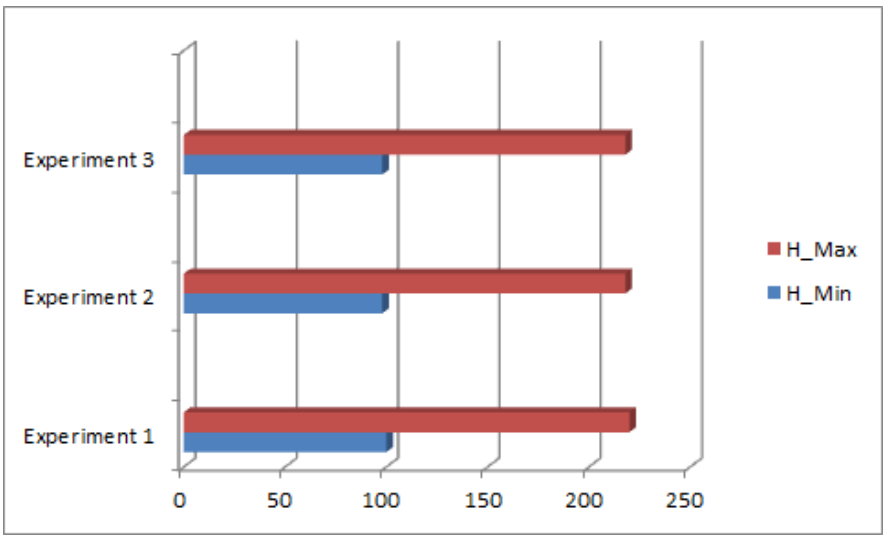

Figure 6-25 Hue range in the three experiments

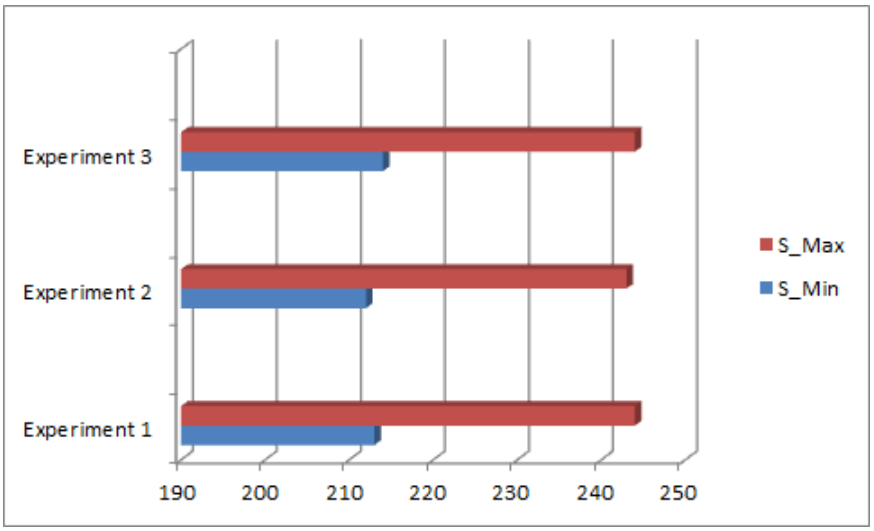

Figure 6-26 Saturation range in the three experiments

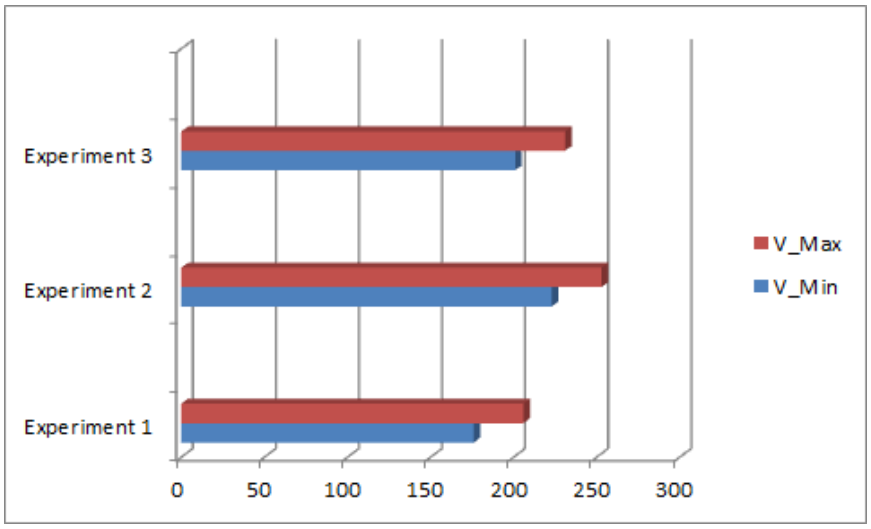

Figure 6-27 Value range in the three experiments

Figure 6-25, Figure 6-26 and Figure 6-27 show graphically that the change in the Value parameter was greater due to the alteration of the brightness/lightness conditions.

Finally and most importantly, after this experiment it is concluded that the circle detection and dynamic color range adjustment method that was developed in this project can effectively detect a new ball under different light conditions.

6.8 SUMMARY

After the testing and the evaluation procedure, the chapter that follows discusses the results and explains at the same time the reason why these methods were developed, concluding finally on the discussion of the successful completion of the objectives of the project.

7 DISCUSSION AND CONCLUSIONS

7.1 DISCUSSION

WowWee's Rovios are telepresence agents whose features and characteristics made the implementations of this project possible. Rovios have been used in several projects (Begum, et al., 2010) mainly because they can wirelessly be controlled through algorithms that are implemented in a PC. In this way the agents can behave in a desired way. Their behaviour can be dynamically adjusted and adapted in unfamiliar environment. As in RoboCup (Asada, et al., 1997), the agents and the target ball changes position while the robots wirelessly interact and co-operate.

In this project Rovios make use of advanced image processing methods in order to fulfil their goals. It is known that the effectiveness of the image processing methods depends heavily on the current light conditions. However, in normal light conditions the agents are able to fulfil their tasks in different and unfamiliar environments. The autonomous ability of the agent to recognise colors in different and complex backgrounds make the completion of difficult tasks, like servoing and navigation scenario, possible.

The need of an efficient and effective GUI will be present even if humans manage to implement totally autonomous agents. Nowadays, this goal is still far. As in this project, agents were able to complete small tasks having a completely robotic autonomous behaviour. Arkin, 1998, suggested that behaviours serve the basic building blocks for robotic actions. Therefore, the GUI that commands or triggers specific robotic behaviour must be easy and user-friendly for the perspective operator. Furthermore, the GUI informs the user or the operator for the condition or the status of the robot and provides useful information for the environment or the objects nearby.

In this concept, crucial characteristic of the GUI is the imageboxes that host the acquired images from the agents. According to the current mission of the robot, useful information is displayed on the image like the execution time (time that each image was processed before been displayed), the current generated auto-command, the position of the target or the frame of the recognised object-blob. For the user or the operator, the images are the main source of information while with them he can inspect the actions of the robot.

The implemented GUI is video-centric. As Drury, et al., 2010, suggests, similarly to our GUI, the video window is the most predominant feature with the most important information located on or around the video screen.

The GUI has two kinds of buttons. There are the buttons that simply command the agent and buttons that trigger specific autonomous behaviours. The combination of these buttons results to a semi-autonomous robotic behaviour. In case of collision, the operator can interfere and help the

agent to resolve some issues. In addition, when the result from the circle detection and the dynamic color range adjustment is not satisfactory, the operator can again adjust properly the color tracking bars for a better and more efficient result.

The developed GUI is parametrical and thus very flexible. Various panels have been created giving the operator the ability to adjust them properly for different results. With the tracking color panel the operator can quickly and effectively find the color range of a desired blob and assign to one of the five available Rovios to approach it in order to have a closer look of the current object. The implemented GUI and the CASTER interface (Kadous, et al., 2006) incorporate different arrangements of small sensor feeds and status readouts placed around the interface.

One of the autonomous behaviours that the GUI can trigger is the navigation and exploration based on the InfaRed sensor that the agent has. The IR sensor is used to avoid possible obstacles and the camera is used just for inspection. Again, here the autonomous behaviour can be transformed in semi-autonomous in case that Rovio gets stuck somewhere by disabling the auto-commanding and for a while help the agent with direct manual commands. The application has the ability to enable two Rovios to navigate around a place at the same time. By adding obstacles on the way of the robots the effectiveness of these methods was successfully tested.

The IR based wandering is the only method that does not use the acquired image as an input. All the other tasks have the acquired images as the main input. An image has a lot of information, although the main difficulty is to make the robot to recognise a part of it and act accordingly.

The edge detection method and its parametric panel give the ability of exploration with higher accuracy. By adjusting the bars, edges and lines are detected that cannot be seen from the original un-processed image. Furthermore, the result showed that the edge detection can be used effectively while the agent is moving.

The color tracking method and its panel is helping the user of the GUI to get familiar with the HSV color format. There are three fixed color ranges, blue, purple and yellow, and another choice where by choosing it the user can search for specific color combinations. The panel can be adjusted with great accuracy and can even detect very small blobs on the incoming image. This feature helps the developer to understand how colors change in different light conditions and as a result how machine vision can effectively be applied for real applications and tasks, like the ones in this project.

In comparison with the suggested technique from Masselli, et al., 2004, we detect the ball based on the color information. Bach, et al., 2005, and the implemented method in this project are taking advantage of the fact that the color of the ball is known.

Implementing a pink color tracking algorithm, the display of the mass centre of the detected pink blob and the segmentation of the image, commanding is auto-generated according to the position of

the displayed mass centre on the image. The number of frames that were processed per second for this method (approximately 5 per second in the GUI) guarantees a robust method for detecting and approaching a pink blob-ball even from a distance.

Bach's color based method succeeds a rate of 25 FPS. The ball detection method of this project succeeds such a high rate of FPS. However, the rate of FPS in the GUI of the ball detection method is significantly decreased mainly because of the multiple threads that are being executed asynchronously.

These implementations have been done in an environment with various shades of yellow. The dominant color of the background is yellow. After implementing the method for detecting and approaching the pink ball, a method for detecting and approaching a yellow ball was developed just by changing the input color range in order to explore the distractions that will be present in a noisier environment. As it was expected, by testing these two methods, it was understood that the rate of success for the pink ball was higher that this for the yellow. It is easier to detect and follow a pink blob in a general yellow background than detect and follow a yellow blob in a yellow background. There were times that the method was running effectively without interferences and there were times that the agent was being misled.

In such conditions the color characteristic is not enough. An object can be described also from its shape, in our case the circularity. This was the main reason that led to the implementation of the circle detection method. In case that the robot is unable to detect the color characteristic, either because of the noisy background or the poor light conditions, the circle detection panel can be adjusted to detect the desired ball (circular object) take the average color of its area in HSV and create dynamically a new HSV color range for detection. Again, at that point the GUI gives the flexibility to the user to correct the HSV values in case that he is not pleased with the result of the automated procedure.

The implemented circle detection method uses the Hough transform. Martins, et al., 2006, proves that the Hough transform is a good method to detect circular shaped objects and is relatively unaffected by image noise.

As in Zhang's (2009) proposed method, the Hough transform is introduced to locate the position of the ball in order to reduce the influence of the complex environment. In this project, the Hough transform is used again to locate the position of the ball, however, it is used in order to sample the color of the detected ball.

The circle detection method was not used for auto-commanding the agent because the circle detection is much slower than the color detection method and also because of its instability. Furthermore, the circle detection method can "lock" an irrelevant object while the color detection

depends only on a good and precise color range setting as an input. The instability of the circle's detection method is being overcome from the color sampling when the desired object is locked.

More complicated scenarios were created after implemented the methods for detecting and approaching the pink and the yellow ball that were based on those basic ones. The visual servoing and navigation scenario is a task that requires higher level of autonomous behaviour. The similarities between the implemented navigation scenario and Begum's, et al., 2010, method are the use of omnidirectional robots and the landmark-based navigation.

Communication among agents has been succeeded through the ball play scenario and the dynamic characteristic of swapping agents during a mission. In the ball play scenario at least two agents have to exchange signals-permissions in order to detect and "kick" the ball. Every time that an agent "kicks" the ball the ball changes position and the other agent has to search, detect and "kick" the ball again. This scenario can involve all the available agents by changing agents sequentially (Rovio 1 will give permission to Rovio 2, Rovio 2 to Rovio 3 and so on).

7.2 Future Research and Improvements

Future research, work and improvements will be presented. These thoughts can be unified under a new postgraduate project that will have the current project as a base and starting point.

The current project focused more in color detection techniques. Therefore, future implementations should explore advanced shape detection techniques in the same way that the parametrical circle detection was implemented. The objective should be not only to detect shapes (squares, triangles, polygons) over clear view but also under different angles.

Combining the color and shape detection methods the agents should be able to detect a greater variety of objects or landmarks for implementing more complex co-operating scenarios. For instance, different weighting will be given to the different colors and shapes, so the agent should recognise and categorise the different objects-landmarks.

The existing parametrical GUI can be upgraded and enriched with these additional characteristics and functionalities.

Commands could be given phonetically. Therefore, the triggering of the missions could be implemented through buttons and sound. Rovios could respond accordingly after each command acknowledgement by playing back responses from the built-in speakers.

Finally, an improvement in human-robot interaction can be implemented and explored by making Rovio able to follow a simple conversation about the recognised objects-landmarks. For example, Rovio could be phonetically commanded to approach the closest object-landmark, respond after its success with a sound message to the operator, after that the operator could ask to give report of the object where the agent should return a sound message reporting its color and its

shape or its inability to generate response. Applying different importance to the various objects, the operator should be able to request the search of the object-landmark with the highest or lowest importance factor.

7.3 CONCLUSIONS

The main aim of this dissertation was to create an autonomous, co-operative telerobotic agent framework for WowWee Rovios. The GUI that was developed concentrates all the functionalities that were applied to the available agents while triggers behaviours that can autonomously be executed even in unfamiliar environments.

The first objective was to explore and study the autonomous behaviour of Rovios. The way that the project was programmed and communicated with these agents applied to them autonomous behaviour by performing desired tasks in an environment without continuous human guidance. Rovios gather information either from their camera or the IR sensor and can work without human intervention for an extended period. Furthermore, the agents can sustain their survival as they can use their base as a charging base.

The second objective was the creation of an interface for the WowWee Rovios to communicate to an external party and/or other Rovios. The GUI that was created allows the operator to manually control the various agents in a variety of ways while inspects them basically through the acquired images. Furthermore, the GUI triggers the execution of specific tasks that require the interaction and the co-operation of the agents for successful completion.

The third objective was to achieve a basic function of autonomous exploration in unfamiliar environments, avoiding obstacles safely. By using the IR sensor, Rovios were commanded in such a way that can navigate around the room avoiding obstacles that they find in their way. Additionally, it is highly feasible having two Rovios exploring the same environment at the same time while their cameras acquire useful information. A more advanced navigation and exploration scenario was created using specific landmarks in the environment that is to be explored.

The fourth objective was to use advanced image processing algorithms to identify and describe objects within unfamiliar environments. Not only advanced image processing algorithms to identify specific shapes and colors were used but also the parametrical GUI provides the flexibility to the operator to adapt Rovios functionalities in dynamic and unfamiliar environments.

The fifth objective was to code a knowledge base to store the robot's experience. The way that the developed framework is programmed and structured enables Rovios to learn and complete a task under different conditions each time. By changing the light conditions or by changing the position of the landmarks, Rovios can still fulfil their goals.

The last objective was to achieve basic communication, and co-operation between at least two robotic agents. The task where two Rovios interact and communicate through their game with the pink ball and the dynamic feature of swapping agents during execution time fulfil this objective.

Overall, this project introduced me to the interesting and challenging areas of Artificial Intelligence, Machine Learning and Robot Vision, and their techniques. Furthermore, the use of Open CV libraries made me appreciate and be familiar with latest advances in image processing methods while programming in an object-oriented language advanced my programming skills using modern tools and programming techniques.

8 BIBLIOGRAPHY

Arkin R. C. Behavior-based Robotics - Cambridge, MA, USA : MIT Press, 1998.

Asada M. [et al.] RoboCup:The Robot World Cup Initiative , Proceedings of the first international conference on Autonomous agents. - New York : ACM Press, 1997. - pp. 340–347.

Ashutosh G. [et al.] Machine Learning in Computer Vision - Springer, 2005.

Bach Joscha and Jüngel Matthias Using pattern matching on a flexible, horizon-aligned grid for robotic vision, LNCS Multilevel Seed Growing Segmentation. - 2005. - p. Springer.

Begum Afroza, Lee Minkyoung and Kim Young J A Simple Visual Servoing and Navigation Algorithm for an Omnidirectional Robot, Human-Centric Computing (HumanCom), 2010 3rd International Conference. - 2010.

Bradski Gary and Kaehler Adrian Learning OpenCV - O'Reilly, 2008.

Canny J. A computational approach to edge detection, IEEE Transactions on Pattern Analysis and machine intelligence 8. - 1986. - pp. 679-714.

Comley R., Sebastian P. and Voon Yap Vooi The Effect of Colour Space on Tracking Robustness, Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference. - 2008.

Drury J [et al.] Improving Human- Robot Interaction through Interface Evolution, 2010. - pp. 183–202.

Drury J. [et al.] Evaluating Human-Robot Interaction in a Search-and-Rescue Context, Proceedings of the Performance Metrics for Intelligent System (PerMIS) Workshop. - 2003.

Farris J. [et al.] Co-evolving Soccer Softbot Team Coordination with Genetic Programming - 1998.

Frintrop S. [et al.] Robust Object Detection at Regions of Interest with an Application in Ball Recognition, Proceedings of the 2005 IEEE International Conference on Robotics and Automation. - Barcelona, Spain : [s.n.], 2005.

Gonzalez Rafael and Woods Richard E Digital Image Processing - [s.l.] : Prentice Hall Press, 2002.

Gopalakrishnan Arati , Greene Sheldon and Sekmen Ali Vision-based Mobile Robot Learning and Navigation, IEEE International Workshop on Robots and Human Interactive Communication. - 2005.

Holly Yanco A. [et al.] Essential Features of Telepresence Robots - 2011.

Itsuki N. Soccer Server: a simulator for RoboCup, In JSAI AI-Symposium. - 1995.

Kadous M. W., Sheh Ka-Man R. and Sammut. C. Effective user interface design for rescue robotics, Proceedings of the ACM/IEEE Conference on Human-Robot Interaction. - 2006.

Kim Whee Kuk and Yi Byung-Ju The Kinematics for Redundantly Actuated Omni-directional Mobile Robots, Proceedings of the 2000 IEEE international Conference on Robotics B Automation. - San Francisco, CA : [s.n.], 2000.

Martins D. A., Neves R. J. and Pinho A. J. Real-time generic ball recognition in RoboCup domain - 2006.

Masselli A., Treptow A. and Zell A. Real-Time Object Tracking for Soccer-Robots without Color Information, Robotics and Autonomous Systems. - [s.l.] : Elsevier, 2004. - pp. 41-48.

Mataric Maja J. Interaction and Intelligent Behavior - 1994.

Newborn Monty Deep Blue: An Artificial Intelligence Milestone - [s.l.] : Springer, 2004.

Nilsson Nils J. Artificial Intelligence: A new Synthesis - San Fransisco, California : Morgan Kaufmann, 1998.

Norvig Peter and Russell Stuart Artificial Intelligence: A Modern Approach. Prenctice - [s.l.] : Prenctice Hall, 1995.

Salustowicz R., Schmidhuber J. and Wiering M. Learning Team Strategies: Soccer Case Studies - 1998.

Stone P. and Veloso M. Layered Approach to Learning Client Behaviors in the Robocup Soccer Server, Applied Artificial Intelligence. - 1998. - pp. 12(2):165–188.

Umbaugh S. E. Computer Vision and Image Processing - [s.l.] : Prentice Hall, 1998.

Zalud Ludek ARGOS System for Heterogeneous Mobile Robot Teleoperation, In Proceedings of IROS'2006. - 2006. - pp. 211-216.

Zhang H., Wu Y. and Yang F. Ball Detection Based on Color Information and Hough Transform, International Conference on Artificial Intelligence and Computational Intelligence. - 2009.

APPENDIX A – WINFORM APPLICATION CODE (VISUAL C#)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.Net;
using System.IO;
using System.Threading;
using System.Runtime.InteropServices;
using Emgu.CV;
using Emgu.CV.UI;
using Emgu.CV.Structure;
using Emgu.Util;
using RovioLib;



namespace rovio_1
{

    public partial class Form1 : Form
    {

                    /*******************************/
                    /*        Public Variables      */
                    /*******************************/


        /*************************************************************************\
        *                                                                         *
        *    Image processing Choices                                             *
        *                                                                         *
        *       0- Default/Normal                                                 *
        *       1- Edge Detection                                                 *
        *       2- Color Detection                                                *
        *       3- Segmenting                                                     *
        *       4- Pink ball commanding (Rovio 2)                                 *
        *       5- Color Tracking                                                 *
        *       6- Pink ball commanding (Rovio 1)                                 *
        *       7- Circle Detection (Rovio 1)                                     *
        *       8- PLAY WITH THE BALL                                             *
        *       9- Yellow Ball commanding (Rovio 1)                               *
        *      10- Yellow Ball commanding (Rovio 2)                               *
        *      11- Mission 1  (pink-> ball->yellow->pink->) for Rovio 1           *
        *      12- New Ball commanding (Rovio 1)                                  *
        *                                                                         *
        \*************************************************************************/

        int processingChoice = 0;

        //Rovio 1 URL
        static string rovio1URL = "http://192.168.2.11";
```

```
        //Rovio 2 URL
        static string rovio2URL = "http://192.168.2.12";
        //Rovio 3 URL
        static string rovio3URL = "http://192.168.2.14";
        //Rovio 4 URL
        static string rovio4URL = "http://192.168.2.15";
        //Rovio 5 URL
        static string rovio5URL = "http://192.168.2.16";



        //UI, declare a delegate:  "display delegates"
        public delegate void MyDelegateMethod(string coordinates);
        public delegate void MyDelegateMethod1();
        public delegate void MyDelegateMethod2();
        public delegate void MyDelegateMethod3(int battery);
        public delegate void MyDelegateMethod4(int battery2);
        public delegate void MyDelegateMethod5(decimal distance);
        public delegate void MyDelegateMethod6();
        public delegate void MyDelegateMethod7(string circlePosition);
        public delegate void MyDelegateMethod8(string color);

        //for circle detection public variables and flags
        Hsv hsv = new Hsv();
        Point centre = new Point();
        bool button20Clicked = false;
        bool button45Clicked = false;

        //set HSV trackbars from automatic color range adjustment
        int hue_min = 0;
        int sat_min = 0;
        int val_min = 0;
        int hue_max = 255;
        int sat_max = 255;
        int val_max = 255;


        //parameters for circle detection
        //number of circles
        int N = 1;
        int thresholdCircle=138;
        int param1 = 11, param2 = 11, param3 = 0, param4 = 0;
        int threshold1 = 1, threshold2 = 3, apertureSize = 5;
        //hough parameters
        int
accResolution=2,minDist=100,cannyThreshold=5,accThreshold=50,minRadius=0,maxRadius=100;


        //edge detection parameters
        int edparam1 = 70, edparam2 = 60;

        //segmentation parameters
        int segments = 5;




        //ir wander 1 thread start/pause
        int lockWander1=0;

        //ir wander 2 thread start/pause
        int lockWander2 = 0;

        //ir wander 1&2 thread start/pause
```

```
        int lockWander = 0;

        //enable flab tab2
        public bool enableTab2 = false;


        /*   Create rovio object for rovio 1 for the first time     */
        RovioController rovio1 = new RovioController("username", "password", rovio1URL);
        /*   Create rovio object for rovio 2  for the first time        */
        RovioController rovio2 = new RovioController("username", "password", rovio2URL);
        /*   Create rovio object for rovio 3  for the first time        */
        RovioController rovio3 = new RovioController("username", "password", rovio3URL);
        /*   Create rovio object for rovio 4  for the first time        */
        RovioController rovio4 = new RovioController("username", "password", rovio4URL);
        /*   Create rovio object for rovio 5  for the first time        */
        RovioController rovio5 = new RovioController("username", "password", rovio5URL);

        //mission 1 flag
        int mission1Flag = 0;

        //go home flag
        int goHomeFlag = 0;

        //play Flag
        int playFlag = 0;

        /*   Initialisation of the Wander Thread        */
        Thread wanderThread1 = new Thread(new ThreadStart(wanderMethod1));
        Thread wanderThread2 = new Thread(new ThreadStart(wanderMethod2));


        /*   Variables for Battery Status  Rovio 1 & 2      */
        public int battery, charging, battery1, charging1;

        /*   Stop/Play flag for wandering Rovio 1& 2       */
        bool wanderFlag = false;
        bool wanderFlag1 = false;
        bool wanderFlag2 = false;

        /*   Integers to hold first character of desired strings from GetReport rovio 1 & 2
*/
        int  firstCharacterCharg, firstCharacterBat1, firstCharacterCharg1;


        /*   Strings from GetReport for rovio 1 & 2    */
        string  str_bat, str_charg, str_bat1, str_charg1;


        /*   Initialisation of rovio's speed   (1-fastest,10-slowest)    */
        public static int speed = 1;


        /*     Positioning Coordinates   */
        public static int pos1x, pos1y, pos2x, pos2y;
        public static decimal pos1theta, pos2theta;

        /*  String Variables for the wander methods    */
        public static string str_ir1, str_ir2;
        public static int flag, firstCharacter1, firstCharacter2;

        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {

            /*   Thread for taking images from Rovio 1      */
            Thread takeImagesThread1 = new Thread(new ThreadStart(takeImages1));
            takeImagesThread1.IsBackground = true;   //for the thread to close with the
application
            takeImagesThread1.Start();
            //takeImagesThread1.Priority = ThreadPriority.Highest;

            /*   Thread for taking images from Rovio 2      */
            Thread takeImagesThread2 = new Thread(new ThreadStart(takeImages2));
            takeImagesThread2.IsBackground = true;    //for the thread to close with the
application
            takeImagesThread2.Start();
            //takeImagesThread.Priority = ThreadPriority.Highest;


            /*   Positioning Thread    */
            Thread positioning_and_collisionAvoidanceThread = new Thread(new
ThreadStart(positioning_and_collisionAvoidance));
            positioning_and_collisionAvoidanceThread.IsBackground = true;
            positioning_and_collisionAvoidanceThread.Start();
            //positioningThread.Priority = ThreadPriority.Lowest;

            /*   Battery Monitoring Thread    */
            Thread batteryMonitoringThread = new Thread(new ThreadStart(batteryMonitoring));
            batteryMonitoringThread.IsBackground = true;
            batteryMonitoringThread.Start();




        }

        /*   Button for Displaying the main getReport  Rovio 1     */
        private void button40_Click(object sender, EventArgs e)
        {

            MessageBox.Show(rovio1.GetReport());
        }

        /*   Button for Displaying the main getReport  Rovio 2     */
        private void button1_Click(object sender, EventArgs e)
        {

            MessageBox.Show(rovio2.GetReport());
        }

        /*   selecting speed of Rovio       */
        private void trackBar10_Scroll(object sender, EventArgs e)
        {
            trackBar10.SetRange(1, 10);
            speed = trackBar10.Value;
        }


        //edge detection first parameter
        private void trackBar1_Scroll(object sender, EventArgs e)
        {

            trackBar1.SetRange(0, 500);
            label17.Text = "Thresh = " + Convert.ToInt32(trackBar1.Value);
```

```
            edparam1 = trackBar1.Value ;
        }

        private void trackBar2_Scroll(object sender, EventArgs e)
        {
            trackBar2.SetRange(0, 500);
            label18.Text = "ThreshLinking = " + Convert.ToInt32(trackBar2.Value);
            edparam2 = trackBar2.Value;
        }

        private void trackBar3_Scroll(object sender, EventArgs e)
        {
            trackBar3.SetRange(2, 10);
            label20.Text = "Segments = " + Convert.ToInt32(trackBar3.Value *
trackBar3.Value);
            segments = trackBar3.Value;
        }


        //min hsv color
        private void trackBar4_Scroll(object sender, EventArgs e)
        {
            trackBar4.SetRange(0, 256);
            label23.Text = "H_MIN = " + Convert.ToInt32(trackBar4.Value);
            hue_min = trackBar4.Value;
        }
        private void trackBar5_Scroll(object sender, EventArgs e)
        {
            trackBar5.SetRange(0, 256);
            label25.Text = "S_MIN = " + Convert.ToInt32(trackBar5.Value);
            sat_min = trackBar5.Value;
        }

        private void trackBar6_Scroll(object sender, EventArgs e)
        {
            trackBar6.SetRange(0, 256);
            label27.Text = "V_MIN = " + Convert.ToInt32(trackBar6.Value);
            val_min = trackBar6.Value;
        }

        //max hsv color
        private void trackBar9_Scroll(object sender, EventArgs e)
        {
            trackBar9.SetRange(0, 256);
            label24.Text = "H_MAX = " + Convert.ToInt32(trackBar9.Value);
            hue_max = trackBar9.Value;
        }

        private void trackBar8_Scroll(object sender, EventArgs e)
        {
            trackBar8.SetRange(0, 256);
            label26.Text = "S_MAX = " + Convert.ToInt32(trackBar8.Value);
            sat_max = trackBar8.Value;
        }

        private void trackBar7_Scroll(object sender, EventArgs e)
        {
            trackBar7.SetRange(0, 256);
            label28.Text = "V_MAX = " + Convert.ToInt32(trackBar7.Value);
            val_max = trackBar7.Value;
        }


        //no of circles detected
```

```
        private void numericUpDown1_ValueChanged(object sender, EventArgs e)
        {
            N = (int)numericUpDown1.Value;
        }

        //threshold in edge detection
        private void trackBar11_Scroll(object sender, EventArgs e)
        {
            trackBar11.SetRange(0, 256);
            thresholdCircle = trackBar11.Value;
            label11.Text = "Threshold = " + Convert.ToInt32(trackBar11.Value);
        }

        //smothing gaussian parameters

        private void numericUpDown2_ValueChanged(object sender, EventArgs e)  //paramet 1
        {
            if(numericUpDown2.Value%2==1)
                param1 = (int)numericUpDown2.Value;
        }

        private void numericUpDown3_ValueChanged(object sender, EventArgs e)   //paramet 2
        {
            if (numericUpDown3.Value % 2 == 1)
                param2 = (int)numericUpDown3.Value;
        }

        private void numericUpDown4_ValueChanged(object sender, EventArgs e)   //paramet 3
        {
            param3 = (int)numericUpDown4.Value;
        }

        private void numericUpDown5_ValueChanged(object sender, EventArgs e) //paramet 2
        {
            param4 = (int)numericUpDown5.Value;
        }

        //canny in circles detection
        private void numericUpDown8_ValueChanged(object sender, EventArgs e)  //threshold 1
        {
            threshold1 = (int)numericUpDown8.Value;
        }

        private void numericUpDown7_ValueChanged(object sender, EventArgs e)   //threshold 2
        {
            threshold2 = (int)numericUpDown7.Value;
        }

        private void numericUpDown6_ValueChanged(object sender, EventArgs e)   //aperture
size
        {
            if (numericUpDown6.Value % 2 == 1)
                apertureSize = (int)numericUpDown6.Value;
        }


        //hough parameters
        private void numericUpDown9_ValueChanged(object sender, EventArgs e)  //acc
resolution
        {
            accResolution = (int)numericUpDown9.Value;
        }

        private void numericUpDown10_ValueChanged(object sender, EventArgs e)  //min
```

```
distance
        {
            minDist =(int)numericUpDown10.Value;
        }

        private void label42_Click(object sender, EventArgs e)  //canny threshold
        {
            cannyThreshold = (int)numericUpDown11.Value;
        }

        private void label43_Click(object sender, EventArgs e)  //accum threshold
        {
            accThreshold = (int)numericUpDown12.Value;
        }

        private void label44_Click(object sender, EventArgs e)   //min radius
        {
            minRadius = (int)numericUpDown13.Value;
        }

        private void label45_Click(object sender, EventArgs e)   //max radius
        {
            maxRadius = (int)numericUpDown14.Value;
        }



                    /////////
                    /////////
                    /////////
        ////////////////////////////////////
        //                                //
        //         Manual Control Rovio 1    //
        //                                //
        ////////////////////////////////////

        private void button31_Click_1(object sender, EventArgs e)  //stop
        {
            rovio1.ManualDrive(0, speed);
        }

        private void button38_Click_1(object sender, EventArgs e)  //forward
        {

            rovio1.ManualDrive(1, speed);
        }

        private void button37_Click_1(object sender, EventArgs e)  //backwards
        {
            rovio1.ManualDrive(2, speed);
        }

        private void button36_Click_1(object sender, EventArgs e)    //right
        {
            rovio1.ManualDrive(4, speed);
        }

        private void button39_Click_1(object sender, EventArgs e)  //left
        {
            rovio1.ManualDrive(3, speed);
        }

        private void button35_Click_1(object sender, EventArgs e)  //forward right
        {
```

```
            rovio1.ManualDrive(8, speed);
        }

        private void button34_Click_1(object sender, EventArgs e)  //forward left
        {
            rovio1.ManualDrive(7, speed);
        }

        private void button32_Click_1(object sender, EventArgs e)   //backwards right
        {
            rovio1.ManualDrive(10, speed);
        }

        private void button33_Click_1(object sender, EventArgs e)   //backwards left
        {
            rovio1.ManualDrive(9, speed);
        }

        private void button30_Click_1(object sender, EventArgs e)  //head Up
        {
            rovio1.ManualDrive(11, speed);
        }

        private void button29_Click_1(object sender, EventArgs e)   //head down
        {
            rovio1.ManualDrive(12, speed);
        }

        private void button28_Click_1(object sender, EventArgs e)  //head middle
        {
            rovio1.ManualDrive(13, speed);
        }

        private void button27_Click_1(object sender, EventArgs e)  //rotate right
        {
            rovio1.ManualDrive(18, speed);
        }

        private void button26_Click_1(object sender, EventArgs e)    //rotate left
        {
            rovio1.ManualDrive(17, speed);
        }


                    /////////
                    /////////
                    /////////
        ////////////////////////////////////
        //                                //
        //         Manual Control Rovio 2    //
        //                                //
        ////////////////////////////////////

        private void button10_Click(object sender, EventArgs e)  //stop
        {
            rovio2.ManualDrive(0, speed);
        }

        private void button2_Click(object sender, EventArgs e)  //forward
        {
                rovio2.ManualDrive(1, speed);
        }

        private void button5_Click(object sender, EventArgs e)  //backwards
```

```
        {
                rovio2.ManualDrive(2, speed);
        }

        private void button4_Click(object sender, EventArgs e)    //right
        {
                rovio2.ManualDrive(4, speed);
        }

        private void button3_Click(object sender, EventArgs e)  //left
        {
                rovio2.ManualDrive(3, speed);
        }

        private void button6_Click(object sender, EventArgs e)  //forward right
        {
            rovio2.ManualDrive(8, speed);
        }

        private void button7_Click(object sender, EventArgs e)  //forward left
        {
            rovio2.ManualDrive(7, speed);
        }

        private void button9_Click(object sender, EventArgs e)   //backwards right
        {
            rovio2.ManualDrive(10, speed);
        }

        private void button8_Click(object sender, EventArgs e)   //backwards left
        {
            rovio2.ManualDrive(9, speed);
        }

        private void button11_Click(object sender, EventArgs e)  //head Up
        {
            rovio2.ManualDrive(11, speed);
        }

        private void button12_Click(object sender, EventArgs e)   //head down
        {
            rovio2.ManualDrive(12, speed);
        }

        private void button13_Click(object sender, EventArgs e)  //head middle
        {
            rovio2.ManualDrive(13, speed);
        }

        private void button14_Click(object sender, EventArgs e)  //rotate right
        {
            rovio2.ManualDrive(18, speed);
        }

        private void button15_Click(object sender, EventArgs e)    //rotate left
        {
            rovio2.ManualDrive(17, speed);
        }

                ////////
                ////////
                ////////
        ////////////////////////////////
        //          (Rovio 1)         //
```

```
        //  IR-based Wandering Method   //
        //                              //
        //              &               //
        //                              //
        //         Button_click         //
        //                              //
        //////////////////////////////////


        private void button42_Click(object sender, EventArgs e)
        {

            if (wanderFlag1 == false && lockWander1==0)
            {
                wanderFlag1 = true;
                //change image to pause
                button42.Image = Image.FromFile(@"C:\Documents and
Settings\kat091\Desktop\Rovio C# Project\Rovio C# Project\rovio 1\images\pause.jpg");
                //activate IR detector
                rovio1.ActivateIRDetector();
                wanderThread1.Start();
                lockWander1 = 1;

            }
            else if (wanderFlag1 == false && lockWander1 == 1 )
            {
                wanderFlag1 = true;
                //change image to pause
                button42.Image = Image.FromFile(@"C:\Documents and
Settings\kat091\Desktop\Rovio C# Project\Rovio C# Project\rovio 1\images\pause.jpg");
                //activate IR detector
                //rovio1.ActivateIRDetector();
                wanderThread1.Resume();
                lockWander1 = 1;
            }
            else if (wanderFlag1 == true)
            {
                wanderFlag1 = false;
                button42.Image = Image.FromFile(@"C:\Documents and
Settings\kat091\Desktop\Rovio C# Project\Rovio C# Project\rovio 1\images\play.jpg");
                //rovio1.DeactivateIRDetector();
                wanderThread1.Suspend();

            }
        }


                ///////////
                ///////////
                ///////////
        //////////////////////////////////
        //            (Rovio 2)         //
        //  IR-based Wandering Method   //
        //                              //
        //              &               //
        //                              //
        //         Button_click         //
        //                              //
        //////////////////////////////////


        //button click method wander Rovio 2
        private void button19_Click(object sender, EventArgs e)
        {
```

```
            if (wanderFlag2 == false && lockWander2==0)
            {
                wanderFlag2 = true;
                //change image to pause
                button19.Image = Image.FromFile(@"C:\Documents and
Settings\kat091\Desktop\Rovio C# Project\Rovio C# Project\rovio 1\images\pause.jpg");
                //activate IR detector
                rovio2.ActivateIRDetector();
                wanderThread2.Start();
                lockWander2 = 1;
            }
            else if (wanderFlag2 == false && lockWander2 == 1)
            {
                wanderFlag2 = true;
                //change image to pause
                button19.Image = Image.FromFile(@"C:\Documents and
Settings\kat091\Desktop\Rovio C# Project\Rovio C# Project\rovio 1\images\pause.jpg");
                //activate IR detector

                wanderThread2.Resume();
                lockWander2 = 1;
            }
            else if (wanderFlag2 == true)
            {
                wanderFlag2 = false;
                button19.Image = Image.FromFile(@"C:\Documents and
Settings\kat091\Desktop\Rovio C# Project\Rovio C# Project\rovio 1\images\play.jpg");
                //rovio2.DeactivateIRDetector();
                wanderThread2.Suspend();
            }


        }



        //wander method for rovio 1 for thread
        public static void wander1()
        {
            RovioController rovio1 = new RovioController("username", "password", rovio1URL);

            while (true)
            {
                str_ir1 = rovio1.GetReport();
                firstCharacter1 = str_ir1.IndexOf("flags");
                str_ir1 = str_ir1.Substring(firstCharacter1 + 9, 1);

                if (str_ir1 != "5")
                {
                    rovio1.ManualDrive(17, speed);  //rotate left
                    rovio1.ManualDrive(0, speed); //stop

                }



            }

        }


        //wander method for rovio 2 for thread
```

```
        public static void wander2()
        {
            RovioController rovio2 = new RovioController("username", "password", rovio2URL);

            while (true)
            {
                str_ir2 = rovio2.GetReport();
                firstCharacter2 = str_ir2.IndexOf("flags");
                str_ir2 = str_ir2.Substring(firstCharacter2 + 9, 1);

                if (str_ir2 != "5")
                {
                    rovio2.ManualDrive(17, speed);  //rotate left
                    rovio2.ManualDrive(0, speed); //stop

                }

            }

        }


        //wander method for rovio 1
        public static void wanderMethod1()
        {
            RovioController rovio1 = new RovioController("username", "password", rovio1URL);

            while (true)
            {
                str_ir1 = rovio1.GetReport();
                firstCharacter1 = str_ir1.IndexOf("flags");
                str_ir1 = str_ir1.Substring(firstCharacter1 + 9, 1);

                if (str_ir1 == "5")
                {
                    rovio1.ManualDrive(1, speed);  //forward

                }
                else
                {
                    rovio1.ManualDrive(17, speed);  //rotate left
                    rovio1.ManualDrive(0, speed); //stop
                }
            }

        }

        //wander method for rovio 2
        public static void wanderMethod2()
        {
            RovioController rovio2 = new RovioController("username", "password", rovio2URL);
            while (true)
            {
                str_ir2 = rovio2.GetReport();
                firstCharacter2 = str_ir2.IndexOf("flags");
                str_ir2 = str_ir2.Substring(firstCharacter2 + 9, 1);

                if (str_ir2 == "5")
                {
                    rovio2.ManualDrive(1, speed);  //forward

                }
                else
                {
```

```
                    rovio2.ManualDrive(17, speed);  //rotate left
                    rovio2.ManualDrive(0, speed); //stop
                }
            }

        }









                 /////////
                /////////
                 /////////
        ////////////////////////////////
        //                            //
        //        Positioning         //
        //            AND             //
        //       collisionAvoidance   //
        //                            //
        ////////////////////////////////
        public void positioning_and_collisionAvoidance()
        {
            decimal previousDistance=0;

            while (true)
            {
                //create the two rovio objects -- IMPORTANCE OF CREATING THE NEW OBJECTS IN
TEH NEW THREAD
                RovioController rovio1 = new RovioController("username", "password",
rovio1URL);
                RovioController rovio2 = new RovioController("username", "password",
rovio2URL);
                //rovio 1 get report
                string strRovio1 = rovio1.GetReport();
                //rovio 2 get report
                string strRovio2 = rovio2.GetReport();

                string str1x = "0";
                string str1y = "0";
                string str1theta = "0";
                string str2x = "0";
                string str2y = "0";
                string str2theta = "0";
                int last = 0;

                //averaging the positions- initialisation
                int avepos1x = 0;
                int avepos1y = 0;
                decimal avepos1theta = 0;
                int avepos2x = 0;
                int avepos2y = 0;
                decimal avepos2theta = 0;

                //averaging the positions- initialisation
                int sumpos1x = 0;
                int sumpos1y = 0;
                decimal sumpos1theta = 0;
                int sumpos2x = 0;
                int sumpos2y = 0;
                decimal sumpos2theta = 0;
```

```
//number of samples taken
int N=10;

for(int i=1;i<=N;i++)
{

    //rovio 1 get report
    strRovio1 = rovio1.GetReport();
    //rovio 2 get report
    strRovio2 = rovio2.GetReport();

    //get the x and y coordinates of the two rovios

    //get Rovio 1 x coordinate

    int first = strRovio1.IndexOf("x=");
    string tempString = strRovio1.Substring(first + 2, 8);
    last = tempString.IndexOf("|");
    str1x = tempString.Substring(0, last);
    pos1x = Convert.ToInt32(str1x);
    sumpos1x += pos1x;

    //get Rovio 1 y coordinate

    first = strRovio1.IndexOf("y=");
    tempString = strRovio1.Substring(first + 2, 8);
    last = tempString.IndexOf("|");
    str1y = tempString.Substring(0, last);
    pos1y = Convert.ToInt32(str1y);
    sumpos1y += pos1y;

    //get Rovio 1 theta coordinate
    first = strRovio1.IndexOf("theta=");
    tempString = strRovio1.Substring(first + 6, 8);
    last = tempString.IndexOf("|");
    str1theta = tempString.Substring(0, last);
    pos1theta = Convert.ToDecimal(str1theta);
    sumpos1theta += pos1theta;

    //get Rovio 2 x coordinate
    first = strRovio2.IndexOf("x=");
    tempString = strRovio2.Substring(first + 2, 8);
    last = tempString.IndexOf("|");
    str2x = tempString.Substring(0, last);
    pos2x = Convert.ToInt32(str2x);
    sumpos2x += pos2x;

    //get Rovio 2 y coordinate

    first = strRovio2.IndexOf("y=");
    tempString = strRovio2.Substring(first + 2, 8);
    last = tempString.IndexOf("|");
    str2y = tempString.Substring(0, last);
    pos2y = Convert.ToInt32(str2y);
    sumpos2y += pos2y;


    //get Rovio 2 theta coordinate
    first = strRovio2.IndexOf("theta=");
    tempString = strRovio2.Substring(first + 6, 8);
    last = tempString.IndexOf("|");
    str2theta = tempString.Substring(0, last);
```

```
                    pos2theta = Convert.ToDecimal(str2theta);
                    sumpos2theta += pos2theta;


                }

                //calculating teh average values
                avepos1x = sumpos1x / N;
                avepos1y = sumpos1y / N;
                avepos1theta = sumpos1theta / N;

                avepos2x = sumpos2x / N;
                avepos2y = sumpos2y / N;
                avepos2theta = (decimal)sumpos2theta / N;


                displayPosRovio1("(" + Convert.ToString(avepos1x) + "," +
Convert.ToString(avepos1y) + ") " + "theta :" +
Convert.ToString(TruncateFunction(pos1theta,2)));
                displayPosRovio2("(" + Convert.ToString(avepos2x) + "," +
Convert.ToString(avepos2y) + ") " + "theta :" + Convert.ToString(TruncateFunction(pos2theta,
2)));

                int xd = avepos2x - avepos1x;
                int yd = avepos2y - avepos1y;

                decimal distance = (decimal) Math.Sqrt(xd * xd + yd * yd);

                //display distance
                displayDistance(TruncateFunction(distance,2));

                //if (distance < 1500)
                //{
                //    if (distance < previousDistance)
                //    {
                //        rovio1.ManualDrive(3, speed);
                //        rovio2.ManualDrive(3, speed);
                //    }
                //    else
                //    {
                //        rovio2.ManualDrive(4, speed);
                //        rovio1.ManualDrive(4, speed);
                //    }
                //}
                previousDistance = distance;

            }
        }



        //battery monitoring method
        public void batteryMonitoring()
        {
            while (true)
            {

                //create the two rovio objects -- IMPORTANCE OF CREATING THE NEW OBJECTS IN
TEH NEW THREAD
                RovioController rovio1 = new RovioController("username", "password",
rovio1URL);
                RovioController rovio2 = new RovioController("username", "password",
rovio2URL);
                //rovio 1 get report
```

```
            string strRovio1 = rovio1.GetReport();
            //rovio 2 get report
            string strRovio2 = rovio2.GetReport();


            int last;

            //battery monitoring
            //1
            str_charg = strRovio2;
            str_charg1 = strRovio1;

            //2
            int firstCharacterBat = str_charg.IndexOf("battery");
            str_bat = str_charg.Substring(firstCharacterBat + 8, 6);
            last = str_bat.IndexOf("|");
            battery = Convert.ToInt32(str_bat.Substring(0, last));

            firstCharacterBat1 = str_charg1.IndexOf("battery");
            str_bat1 = str_charg1.Substring(firstCharacterBat1 + 8, 6);
            last = str_bat1.IndexOf("|");
            battery1 = Convert.ToInt32(str_bat1.Substring(0, last));

            //3
            firstCharacterCharg = str_charg.IndexOf("charging");
            str_charg = str_charg.Substring(firstCharacterCharg + 9, 2);

            firstCharacterCharg1 = str_charg1.IndexOf("charging");
            str_charg1 = str_charg1.Substring(firstCharacterCharg1 + 9, 2);

            //4
            if (str_charg == "80")
            {
                rollProgressBar2();
            }
            else
            {
                displayBar2(battery);
            }

            if (str_charg1 == "80")
            {
                rollProgressBar1();
            }
            else
            {
                displayBar1(battery1);
            }



        }
    }



            /////////
            /////////
            /////////
    ////////////////////////////////
    //                            //
    //take images method for Rovio 1//
    //                            //
```

```
        ////////////////////////////////


        public void takeImages1()
        {

            while (true)
            {

                //create stopwatch start in the beginning of the thread execution
                Stopwatch sw = new Stopwatch();
                sw.Start();

                //url string command initialisation
                string sourceURL = rovio1URL + "//Jpeg/CamImg.jpg";
                //pictureBox1.Load(sourceURL);
                byte[] buffer = new byte[100000];
                int read, total = 0;

                // create HTTP request
                HttpWebRequest req = (HttpWebRequest)WebRequest.Create(sourceURL);

                // get response
                WebResponse resp = req.GetResponse();

                // get response stream
                Stream stream = resp.GetResponseStream();

                // read data from stream
                while ((read = stream.Read(buffer, total, 1000)) != 0)
                {
                    total += read;
                }

                // get bitmap
                Bitmap bmp = (Bitmap)Bitmap.FromStream(new MemoryStream(buffer, 0, total));

                Image<Bgr, Byte> frame = new Image<Bgr, Byte>((Bitmap)bmp);
                Image<Bgr, Byte> frameOutput = frame;
                Image<Gray, Byte> frameOutputGray = frame.Convert<Gray, Byte>();


                if (processingChoice == 1)
                {

                    frameOutputGray = EdgeDetection(frame);
                    imageBox1.Image = displayExecutionTimeInGray(frameOutputGray,sw);

                }
                else if (processingChoice == 2)
                {
                    frameOutput = PinkTracking(frame);
                    imageBox1.Image = displayExecutionTimeInColor(frameOutput, sw);
                }
                else if (processingChoice == 3)
                {
                    frameOutput = Segmenting(frame);
                    imageBox1.Image = displayExecutionTimeInColor(frameOutput, sw);
                }
                else if (processingChoice == 6 || processingChoice == 8 ||
processingChoice==11)
                {
                    frameOutput = PinkBallCommanding1(frame);
                    imageBox1.Image = displayExecutionTimeInColor(frameOutput, sw);
```

```
                }
                else if (processingChoice == 5)
                {
                    //color range initialisation
                    MCvScalar hsv_min = new MCvScalar(0, 0, 0);
                    MCvScalar hsv_max = new MCvScalar(0, 0, 0);

                    if (checkBox1.Checked)
                    {
                        //range for blue color
                        hsv_min = new MCvScalar(110, 50, 110);
                        hsv_max = new MCvScalar(124, 180, 200);

                    }
                    else if (checkBox2.Checked)
                    {
                        //range for purple color
                        hsv_min = new MCvScalar(125, 50, 110);
                        hsv_max = new MCvScalar(150, 180, 200);
                    }
                    else if (checkBox3.Checked)
                    {
                        //range for yellow color
                        hsv_min = new MCvScalar(30, 101, 102);
                        hsv_max = new MCvScalar(53, 174, 239);
                    }
                    else if (checkBox4.Checked)
                    {
                        //range for ? color
                        hsv_min = new MCvScalar(hue_min, sat_min, val_min);
                        hsv_max = new MCvScalar(hue_max, sat_max, val_max);
                    }
                    else
                    {
                        //black color
                        hsv_min = new MCvScalar(0, 0, 0);
                        hsv_max = new MCvScalar(0, 0, 0);
                    }

                    frameOutputGray = colorTracking(frame, hsv_min, hsv_max);
                    imageBox1.Image = displayExecutionTimeInGray(frameOutputGray, sw);
                }
                else if (processingChoice == 9)
                {
                    frameOutput = YellowBallCommanding1(frame);
                    imageBox1.Image = displayExecutionTimeInColor(frameOutput, sw);
                }
                else if (processingChoice == 7)
                {
                    frameOutput = CircleAndColorDetection(frame);
                    imageBox1.Image = displayExecutionTimeInColor(frameOutput, sw);
                }
                else if (processingChoice == 12)
                {
                    frameOutput = NewBallCommanding1(frame);
                    imageBox1.Image = displayExecutionTimeInColor(frameOutput, sw);
                }
                else  //Normal
                {
                    //display Image in normal mode
                    imageBox1.Image = displayExecutionTimeInColor(frameOutput, sw);

                }
                //standard for tab 2
```

```
                if (enableTab2 == true)
                {
                    imageBox2.Image = displayExecutionTimeInColor(frame, sw);
                }
                //end stopwatch -calculate duration of each frame display
                sw.Stop();

            }
        }



                ////////
                ////////
                ////////
        ////////////////////////////////
        //                            //
        //take images method for Rovio 2//
        //                            //
        ////////////////////////////////

        public void takeImages2()
        {

            while (true)
            {
                //create stopwatch start in the beginning of the thread execution
                Stopwatch sw = new Stopwatch();
                sw.Start();

                //url string command initialisation
                string sourceURL = rovio2URL + "//Jpeg/CamImg.jpg";
                //pictureBox1.Load(sourceURL);
                byte[] buffer = new byte[100000];
                int read, total = 0;

                // create HTTP request
                HttpWebRequest req = (HttpWebRequest)WebRequest.Create(sourceURL);

                // get response
                WebResponse resp = req.GetResponse();

                // get response stream
                Stream stream = resp.GetResponseStream();

                // read data from stream
                while ((read = stream.Read(buffer, total, 1000)) != 0)
                {
                    total += read;
                }
                // get bitmap

                Bitmap bmp = (Bitmap)Bitmap.FromStream(new MemoryStream(buffer, 0, total));

                Image<Bgr, Byte> frame = new Image<Bgr, Byte>((Bitmap)bmp);
                Image<Bgr, Byte> frameOutput = frame;
                Image<Gray, Byte> frameOutputGray = frame.Convert<Gray, Byte>();

                if (processingChoice == 1)
                {
                    frameOutputGray = EdgeDetection(frame);
                    captureImageBox.Image = displayExecutionTimeInGray(frameOutputGray, sw);
;
                }
```

```
                else if (processingChoice == 2)
                {
                    frameOutput = PinkTracking(frame);
                    captureImageBox.Image = displayExecutionTimeInColor(frameOutput, sw);
                }
                else if (processingChoice == 3)
                {
                    frameOutput = Segmenting(frame);
                    captureImageBox.Image = displayExecutionTimeInColor(frameOutput, sw);
                }
                else if (processingChoice == 4)   //originally 4
                {
                    frameOutput = PinkBallCommanding2(frame);
                    captureImageBox.Image = displayExecutionTimeInColor(frameOutput, sw);
                }
                else if (processingChoice == 5)
                {
                    //color range initialisation
                    MCvScalar hsv_min = new MCvScalar(0, 0, 0);
                    MCvScalar hsv_max = new MCvScalar(0, 0, 0);

                    if (checkBox1.Checked)
                    {
                        //range for blue color
                        hsv_min = new MCvScalar(110, 50, 110);
                        hsv_max = new MCvScalar(124, 180, 200);
                    }
                    else if (checkBox2.Checked)
                    {
                        //range for purple color
                        hsv_min = new MCvScalar(125, 50, 110);
                        hsv_max = new MCvScalar(150, 180, 200);
                    }
                    else if (checkBox3.Checked)
                    {
                        //range for yellow color
                        hsv_min = new MCvScalar(30, 101, 102);
                        hsv_max = new MCvScalar(53, 174, 239);
                    }
                    else if (checkBox4.Checked)
                    {
                        //range for ? color
                        hsv_min = new MCvScalar(hue_min, sat_min, val_min);
                        hsv_max = new MCvScalar(hue_max, sat_max, val_max);
                    }
                    else
                    {
                        //black color
                        hsv_min = new MCvScalar(0, 0, 0);
                        hsv_max = new MCvScalar(0, 0, 0);
                    }

                    frameOutputGray = colorTracking(frame, hsv_min, hsv_max);

                    captureImageBox.Image = displayExecutionTimeInGray(frameOutputGray, sw);
                }
                else if (processingChoice == 10)
                {
                    frameOutput = YellowBallCommanding2(frame);
                    captureImageBox.Image = displayExecutionTimeInColor(frameOutput, sw);
                }
                else  //Normal
                {
                    //display Image in normal mode
```

```
                captureImageBox.Image = displayExecutionTimeInColor(frameOutput, sw);

            }

            //tab 2 display
            if (enableTab2 == true)
            {
                imageBox3.Image = displayExecutionTimeInColor(frame, sw);
            }
            //end stopwatch -calculate duration of each frame display
            sw.Stop();


        }


    }

    /////////
    /////////
    /////////
    ////////////////////////////////
    //                            //
    //take images method for Rovio 3//
    //                            //
    ////////////////////////////////


    public void takeImages3()
    {

        while (true)
        {

            //create stopwatch start in the beginning of the thread execution
            Stopwatch sw = new Stopwatch();
            sw.Start();

            //url string command initialisation
            string sourceURL = rovio3URL + "//Jpeg/CamImg.jpg";
            //pictureBox1.Load(sourceURL);
            byte[] buffer = new byte[100000];
            int read, total = 0;

            // create HTTP request
            HttpWebRequest req = (HttpWebRequest)WebRequest.Create(sourceURL);

            // get response
            WebResponse resp = req.GetResponse();

            // get response stream
            Stream stream = resp.GetResponseStream();

            // read data from stream
            while ((read = stream.Read(buffer, total, 1000)) != 0)
            {
                total += read;
            }

            // get bitmap
            Bitmap bmp = (Bitmap)Bitmap.FromStream(new MemoryStream(buffer, 0, total));

            Image<Bgr, Byte> frame = new Image<Bgr, Byte>((Bitmap)bmp);
            Image<Bgr, Byte> frameOutput = frame;
```

```
            Image<Gray, Byte> frameOutputGray = frame.Convert<Gray, Byte>();



             //display Image in normal mode

            imageBox4.Image = displayExecutionTimeInColor(frameOutput, sw);


            //end stopwatch -calculate duration of each frame display
            sw.Stop();
        }
    }


    /////////
    /////////
    /////////
    ////////////////////////////////
    //                            //
    //take images method for Rovio 4//
    //                            //
    ////////////////////////////////


    public void takeImages4()
    {

        while (true)
        {

            //create stopwatch start in the beginning of the thread execution
            Stopwatch sw = new Stopwatch();
            sw.Start();

            //url string command initialisation
            string sourceURL = rovio4URL + "//Jpeg/CamImg.jpg";
            //pictureBox1.Load(sourceURL);
            byte[] buffer = new byte[100000];
            int read, total = 0;

            // create HTTP request
            HttpWebRequest req = (HttpWebRequest)WebRequest.Create(sourceURL);

            // get response
            WebResponse resp = req.GetResponse();

            // get response stream
            Stream stream = resp.GetResponseStream();

            // read data from stream
            while ((read = stream.Read(buffer, total, 1000)) != 0)
            {
                total += read;
            }

            // get bitmap
            Bitmap bmp = (Bitmap)Bitmap.FromStream(new MemoryStream(buffer, 0, total));

            Image<Bgr, Byte> frame = new Image<Bgr, Byte>((Bitmap)bmp);
            Image<Bgr, Byte> frameOutput = frame;
            Image<Gray, Byte> frameOutputGray = frame.Convert<Gray, Byte>();
```

```
            //display Image in normal mode

            imageBox5.Image = displayExecutionTimeInColor(frameOutput, sw);


            //end stopwatch -calculate duration of each frame display
            sw.Stop();
        }
    }


    /////////
    /////////
    /////////
    ////////////////////////////////
    //                            //
    //take images method for Rovio 5//
    //                            //
    ////////////////////////////////


    public void takeImages5()
    {

        while (true)
        {

            //create stopwatch start in the beginning of the thread execution
            Stopwatch sw = new Stopwatch();
            sw.Start();

            //url string command initialisation
            string sourceURL = rovio5URL + "//Jpeg/CamImg.jpg";
            //pictureBox1.Load(sourceURL);
            byte[] buffer = new byte[100000];
            int read, total = 0;

            // create HTTP request
            HttpWebRequest req = (HttpWebRequest)WebRequest.Create(sourceURL);

            // get response
            WebResponse resp = req.GetResponse();

            // get response stream
            Stream stream = resp.GetResponseStream();

            // read data from stream
            while ((read = stream.Read(buffer, total, 1000)) != 0)
            {
                total += read;
            }

            // get bitmap
            Bitmap bmp = (Bitmap)Bitmap.FromStream(new MemoryStream(buffer, 0, total));

            Image<Bgr, Byte> frame = new Image<Bgr, Byte>((Bitmap)bmp);
            Image<Bgr, Byte> frameOutput = frame;
            Image<Gray, Byte> frameOutputGray = frame.Convert<Gray, Byte>();



            //display Image in normal mode
```

```
                imageBox6.Image = displayExecutionTimeInColor(frameOutput, sw);


                //end stopwatch -calculate duration of each frame display
                sw.Stop();
            }
        }


        /* Go home Button and command  - Rovio 1   */
        private void button48_Click(object sender, EventArgs e)
        {
            rovio1.GoHomeAndDock();
        }


        private void button22_Click(object sender, EventArgs e)
        {
            rovio2.GoHomeAndDock();
        }

        /*    form closing    */
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (wanderThread2.IsAlive == true)
            {
                wanderThread2.Abort();

            }
            if (wanderThread1.IsAlive == true)
            {
                wanderThread2.Abort();

            }




        }



        ////////////////////////////////////////////////////////////////
        //                                                            //
        //          Image Processing Routines using OpenCV            //
        //                                                            //
        ////////////////////////////////////////////////////////////////

                    /*************************************/
                    /*          Edge Detection           */
                    /*************************************/


       public Image<Gray,Byte> EdgeDetection(Image<Bgr,Byte> image)
       {
            Image<Gray, Byte> grayFrame = image.Convert<Gray, Byte>();
            Image<Gray, Byte> smallGrayFrame = grayFrame.PyrDown();
            Image<Gray, Byte> smoothedGrayFrame = smallGrayFrame.PyrUp();
            //Image<Gray, Byte> cannyFrame = smoothedGrayFrame.Canny(new Gray(100), new
Gray(60));   //100,60
            Image<Gray, Byte> cannyFrame = smoothedGrayFrame.Canny(new Gray(edparam1), new
Gray(edparam2));
            return cannyFrame;
       }
```

```
                    /************************************/
                    /*         Tracking Pink Color      */
                    /************************************/


        public Image<Bgr, Byte> PinkTracking(Image<Bgr, Byte> image)
        {



            MCvMoments moments = new MCvMoments();

            //MCvScalar hsv_min=new MCvScalar(0,50,170);
            //MCvScalar hsv_max=new MCvScalar(10,180,256);
            //MCvScalar hsv_min2=new MCvScalar(170,50,170);
            //MCvScalar hsv_max2=new MCvScalar(256,180,256);

            MCvScalar hsv_min = new MCvScalar(172, 147, 185);
            MCvScalar hsv_max = new MCvScalar(176, 211, 256);

            Image<Gray, Byte> thresholded = image.Convert<Gray, Byte>();
            Image<Gray, Byte> thresholded2 = image.Convert<Gray, Byte>();
            Image<Hsv, Byte> hsv_image = image.Convert<Hsv, Byte>();

            CvInvoke.cvCvtColor(image, hsv_image,
Emgu.CV.CvEnum.COLOR_CONVERSION.CV_BGR2HSV);
            CvInvoke.cvInRangeS(hsv_image, hsv_min, hsv_max, thresholded);
            //CvInvoke.cvInRangeS(hsv_image, hsv_min2, hsv_max2, thresholded2);
            //CvInvoke.cvOr(thresholded, thresholded2, thresholded,IntPtr.Zero);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvThreshold(thresholded, thresholded, 12, 256,
Emgu.CV.CvEnum.THRESH.CV_THRESH_BINARY);
            int iterations = 5;
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);

            CvInvoke.cvMoments(thresholded, ref moments, 1);

            double moment10 = CvInvoke.cvGetSpatialMoment(ref moments, 1, 0);
            double moment01 = CvInvoke.cvGetSpatialMoment(ref moments, 0, 1);
            double areaPink = CvInvoke.cvGetCentralMoment(ref moments, 0, 0);

            int posX = 0;
            int posY = 0;

            //exception handling
            try
            {
                posX = Convert.ToInt32(moment10 / areaPink);
                posY = Convert.ToInt32(moment01 / areaPink);
            }
            catch
            {
                //do nothing
            }
```

```
            Point center=new Point(posX,posY);

            MCvScalar colorCenter=new MCvScalar(204,102,255);
            MCvFont font = new MCvFont();
            MCvScalar colorFont=new MCvScalar(0,0,255);


            if (posX > 0 && posY > 0)
            {

                CvInvoke.cvCircle(image, center, 10, colorCenter, -1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                CvInvoke.cvInitFont(ref font,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX_SMALL, 1, 1, 0, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
                string textCenter = "(" + Convert.ToString(center.X) + "," +
Convert.ToString(center.Y) + ")";
                CvInvoke.cvPutText(image, textCenter, center, ref font, colorFont);
            }

            return image;
        }


                        /************************************/
                        /*          Segmenting Image        */
                        /************************************/

        public Image<Bgr, Byte> Segmenting(Image<Bgr, Byte> image)
        {



            MCvMoments moments = new MCvMoments();

            MCvScalar hsv_min = new MCvScalar(0, 50, 170);
            MCvScalar hsv_max = new MCvScalar(10, 180, 256);
            MCvScalar hsv_min2 = new MCvScalar(170, 50, 170);
            MCvScalar hsv_max2 = new MCvScalar(256, 180, 256);


            Image<Gray, Byte> thresholded = image.Convert<Gray, Byte>();
            Image<Gray, Byte> thresholded2 = image.Convert<Gray, Byte>();
            Image<Hsv, Byte> hsv_image = image.Convert<Hsv, Byte>();

            CvInvoke.cvCvtColor(image, hsv_image,
Emgu.CV.CvEnum.COLOR_CONVERSION.CV_BGR2HSV);
            CvInvoke.cvInRangeS(hsv_image, hsv_min, hsv_max, thresholded);
            CvInvoke.cvInRangeS(hsv_image, hsv_min2, hsv_max2, thresholded2);
            CvInvoke.cvOr(thresholded, thresholded2, thresholded, IntPtr.Zero);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvThreshold(thresholded, thresholded, 12, 256,
Emgu.CV.CvEnum.THRESH.CV_THRESH_BINARY);
            int iterations = 5;
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
```

```
            CvInvoke.cvMoments(thresholded, ref moments, 1);

            double moment10 = CvInvoke.cvGetSpatialMoment(ref moments, 1, 0);
            double moment01 = CvInvoke.cvGetSpatialMoment(ref moments, 0, 1);
            double areaPink = CvInvoke.cvGetCentralMoment(ref moments, 0, 0);

            int posX = 0;
            int posY = 0;

            //exception handling
            try
            {
                posX = Convert.ToInt32(moment10 / areaPink);
                posY = Convert.ToInt32(moment01 / areaPink);
            }
            catch
            {
                //do nothing
            }
            Point center = new Point(posX, posY);
            MCvScalar colorCenter = new MCvScalar(204, 102, 255);
            MCvFont font = new MCvFont();
            MCvScalar colorFont = new MCvScalar(0, 0, 255);


            if (posX > 0 && posY > 0)
            {

                CvInvoke.cvCircle(image, center, 10, colorCenter, -1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                CvInvoke.cvInitFont(ref font,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX_SMALL, 1, 1, 0, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
                string textCenter = "(" + Convert.ToString(center.X) + "," +
Convert.ToString(center.Y) + ")";
                CvInvoke.cvPutText(image, textCenter, center, ref font, colorFont);
            }

            //1st segment
            MCvScalar colorRect = new MCvScalar(0, 255, 255);
            Point p1 = new Point(image.Width / 5, image.Height / 5);
            Point p2 = new Point(image.Width / 5, image.Height / 5);

            for (int i = 0; i <= segments; i++)
            {
                for (int j = 0; j <= segments; j++)
                {
                    //create points
                    p1 = new Point(i * image.Width / segments, j * image.Height / segments);
                    p2 = new Point((i + 1) * image.Width / segments, (j + 1) * image.Height /
segments);

                    //display points
                    MCvFont font1 = new MCvFont();
                    CvInvoke.cvInitFont(ref font1,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX,0.5f,0.5f, 0,1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
                    string textRectPointsP1 = "(" + Convert.ToString(p1.X) + "," +
Convert.ToString(p1.Y) + ")";
                    string textRectPointsP2 = "(" + Convert.ToString(p2.X) + "," +
Convert.ToString(p2.Y) + ")";
                    CvInvoke.cvPutText(image, textRectPointsP1, p1, ref font1, colorRect);
                    CvInvoke.cvPutText(image, textRectPointsP2, p2, ref font1, colorRect);
```

```
                //display rectangles
                CvInvoke.cvRectangle(image, p1, p2, colorRect, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
            }
        }
        return image;
    }

    /*****************************************/
    /*          Pink Ball Commanding  Rovio 1   */
    /*****************************************/


    public Image<Bgr, Byte> PinkBallCommanding1(Image<Bgr, Byte> image)
    {

        MCvMoments moments = new MCvMoments();

        //MCvScalar hsv_min = new MCvScalar(0, 50, 170);
        //MCvScalar hsv_max = new MCvScalar(10, 180, 256);
        //MCvScalar hsv_min2 = new MCvScalar(170, 50, 170);
        //MCvScalar hsv_max2 = new MCvScalar(256, 180, 256);

        MCvScalar hsv_min = new MCvScalar(172, 147, 185);
        MCvScalar hsv_max = new MCvScalar(176, 211, 256);

        Image<Gray, Byte> thresholded = image.Convert<Gray, Byte>();
        Image<Gray, Byte> thresholded2 = image.Convert<Gray, Byte>();
        Image<Hsv, Byte> hsv_image = image.Convert<Hsv, Byte>();

        CvInvoke.cvCvtColor(image, hsv_image,
Emgu.CV.CvEnum.COLOR_CONVERSION.CV_BGR2HSV);
        CvInvoke.cvInRangeS(hsv_image, hsv_min, hsv_max, thresholded);
        //CvInvoke.cvInRangeS(hsv_image, hsv_min2, hsv_max2, thresholded2);
        //CvInvoke.cvOr(thresholded, thresholded2, thresholded, IntPtr.Zero);
        CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
        CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
        CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
        CvInvoke.cvThreshold(thresholded, thresholded, 12, 256,
Emgu.CV.CvEnum.THRESH.CV_THRESH_BINARY);
        int iterations = 5;
        CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
        CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
        CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
        CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);


        CvInvoke.cvMoments(thresholded, ref moments, 1);

        double moment10 = CvInvoke.cvGetSpatialMoment(ref moments, 1, 0);
        double moment01 = CvInvoke.cvGetSpatialMoment(ref moments, 0, 1);
        double areaPink = CvInvoke.cvGetCentralMoment(ref moments, 0, 0);

        int posX = 0;
        int posY = 0;

        //exception handling
        try
        {
            posX = Convert.ToInt32(moment10 / areaPink);
```

```
                posY = Convert.ToInt32(moment01 / areaPink);
            }
            catch
            {
                //do nothing
            }
            Point center = new Point(posX, posY);
            MCvScalar colorCenter = new MCvScalar(0, 0, 255);
            MCvFont font = new MCvFont();
            MCvScalar colorFont = new MCvScalar(255, 0, 0);


            if (posX > 0 && posY > 0)
            {
                // draw circle around the center of the detected pink blob
                CvInvoke.cvCircle(image, center, 10, colorCenter, -1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                CvInvoke.cvInitFont(ref font,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX_SMALL, 1, 1, 0, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
                string textCenter = "(" + Convert.ToString(center.X) + "," +
Convert.ToString(center.Y) + ")";
                CvInvoke.cvPutText(image, textCenter, center, ref font, colorFont);
            }


            //Action message point and message color
            Point pointActionMessage = new Point(0, 30);
            MCvScalar colorMessage = new MCvScalar(0, 255, 255);
            string textStop = "Action:Stop";
            string textForwardRight = "Action:Forward Right";
            string textForwardLeft = "Action:Forward Left";
            string textForward = "Action:Forward";
            string textRotateLeft = "Action:Rotate Left";




            //searching and centralizing algorithm 6
            if (center.Y > 440) //440 originally
            {
                if (goHomeFlag == 1)
                {
                    rovio1.GoHome();
                    processingChoice = 0;
                    goHomeFlag = 0;
                }
                if (processingChoice == 11)
                {
                    processingChoice = 9;//yellow ball
                    mission1Flag = 1;
                }
                if (processingChoice == 8)   //play with the ball
                {
                    //give permission to rovio 2
                    processingChoice = 4;
                    playFlag = 1;
                }
                //stop Rovio 2
                rovio1.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);
```

```
            }
            else if (center.Y > 96)
                if (center.X > 512)
                {
                    rovio1.ManualDrive(8, speed);  //forward right
                    CvInvoke.cvPutText(image, textForwardRight, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 348)
                {

                    rovio1.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward , pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 256)
                {
                    rovio1.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 128)
                {
                    rovio1.ManualDrive(1, speed);  //forward
                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else
                {
                    rovio1.ManualDrive(7, speed);  //forward left
                    CvInvoke.cvPutText(image, textForwardLeft, pointActionMessage, ref font,
colorMessage);
                }
            else
            {
                rovio1.ManualDrive(17, speed); //rotate left
                CvInvoke.cvPutText(image, textRotateLeft, pointActionMessage, ref font,
colorMessage);
                rovio1.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);
            }



            return image;
        }

                            /*****************************************/
                            /*          Pink Ball Commanding  Rovio 2   */
                            /*****************************************/


        public Image<Bgr, Byte> PinkBallCommanding2(Image<Bgr, Byte> image)
        {

            MCvMoments moments = new MCvMoments();

            //MCvScalar hsv_min = new MCvScalar(0, 50, 170);
            //MCvScalar hsv_max = new MCvScalar(10, 180, 256);
```

```
            //MCvScalar hsv_min2 = new MCvScalar(170, 50, 170);
            //MCvScalar hsv_max2 = new MCvScalar(256, 180, 256);

            MCvScalar hsv_min = new MCvScalar(172, 147, 185);
            MCvScalar hsv_max = new MCvScalar(176, 211, 256);


            Image<Gray, Byte> thresholded = image.Convert<Gray, Byte>();
            Image<Gray, Byte> thresholded2 = image.Convert<Gray, Byte>();
            Image<Hsv, Byte> hsv_image = image.Convert<Hsv, Byte>();

            CvInvoke.cvCvtColor(image, hsv_image,
Emgu.CV.CvEnum.COLOR_CONVERSION.CV_BGR2HSV);
            CvInvoke.cvInRangeS(hsv_image, hsv_min, hsv_max, thresholded);
            //CvInvoke.cvInRangeS(hsv_image, hsv_min2, hsv_max2, thresholded2);
            //CvInvoke.cvOr(thresholded, thresholded2, thresholded, IntPtr.Zero);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvThreshold(thresholded, thresholded, 12, 256,
Emgu.CV.CvEnum.THRESH.CV_THRESH_BINARY);
            int iterations = 5;
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);


            CvInvoke.cvMoments(thresholded, ref moments, 1);

            double moment10 = CvInvoke.cvGetSpatialMoment(ref moments, 1, 0);
            double moment01 = CvInvoke.cvGetSpatialMoment(ref moments, 0, 1);
            double areaPink = CvInvoke.cvGetCentralMoment(ref moments, 0, 0);

            int posX = 0;
            int posY = 0;

            //exception handling
            try
            {
                posX = Convert.ToInt32(moment10 / areaPink);
                posY = Convert.ToInt32(moment01 / areaPink);
            }
            catch
            {
                //do nothing
            }
            Point center = new Point(posX, posY);
            MCvScalar colorCenter = new MCvScalar(0, 0, 255);
            MCvFont font = new MCvFont();
            MCvScalar colorFont = new MCvScalar(255, 0, 0);


            if (posX > 0 && posY > 0)
            {
                // draw circle around the center of the detected pink blob
                CvInvoke.cvCircle(image, center, 10, colorCenter, -1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                CvInvoke.cvInitFont(ref font,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX_SMALL, 1, 1, 0, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
```

```
                string textCenter = "(" + Convert.ToString(center.X) + "," +
Convert.ToString(center.Y) + ")";
                CvInvoke.cvPutText(image, textCenter, center, ref font, colorFont);
            }


            //Action message point and message color
            Point pointActionMessage=new Point(0,30);
            MCvScalar colorMessage = new MCvScalar(0, 255, 255);
            string textStop = "Action:Stop";
            string textForwardRight = "Action:Forward Right";
            string textForwardLeft = "Action:Forward Left";
            string textForward = "Action:Forward";
            string textRotateLeft = "Action:Rotate Left";

            //searching and centralizing algorithm 6
            if (center.Y > 440) //originally 440
            {


                if (playFlag == 1)
                {
                    processingChoice = 8;
                }

                //stop Rovio 2
                rovio2.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);

            }
            else if (center.Y > 96)
                if (center.X > 512)
                {
                    rovio2.ManualDrive(8, speed);  //forward right
                    CvInvoke.cvPutText(image, textForwardRight, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 348)
                {
                    rovio2.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 256)
                {
                    rovio2.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 128)
                {

                    rovio2.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else
                {
                    rovio2.ManualDrive(7, speed); //forward left
```

```
                    CvInvoke.cvPutText(image, textForwardLeft, pointActionMessage, ref font,
colorMessage);
                }
            else
            {
                rovio2.ManualDrive(17, speed); //rotate left
                CvInvoke.cvPutText(image, textRotateLeft, pointActionMessage, ref font,
colorMessage);
                rovio2.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);
            }



            return image;
        }


        /*******************************************/
        /*          Yellow Ball Commanding  Rovio 1   */
        /*******************************************/


        public Image<Bgr, Byte> YellowBallCommanding1(Image<Bgr, Byte> image)
        {

            MCvMoments moments = new MCvMoments();

            MCvScalar hsv_min = new MCvScalar(30, 101, 102);
            MCvScalar hsv_max = new MCvScalar(53, 174, 239);



            Image<Gray, Byte> thresholded = image.Convert<Gray, Byte>();

            Image<Hsv, Byte> hsv_image = image.Convert<Hsv, Byte>();

            CvInvoke.cvCvtColor(image, hsv_image,
Emgu.CV.CvEnum.COLOR_CONVERSION.CV_BGR2HSV);
            CvInvoke.cvInRangeS(hsv_image, hsv_min, hsv_max, thresholded);

            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvThreshold(thresholded, thresholded, 12, 256,
Emgu.CV.CvEnum.THRESH.CV_THRESH_BINARY);
            int iterations = 5;
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);



            CvInvoke.cvMoments(thresholded, ref moments, 1);

            double moment10 = CvInvoke.cvGetSpatialMoment(ref moments, 1, 0);
            double moment01 = CvInvoke.cvGetSpatialMoment(ref moments, 0, 1);
            double areaPink = CvInvoke.cvGetCentralMoment(ref moments, 0, 0);

            int posX = 0;
```

```
            int posY = 0;

            //exception handling
            try
            {
                posX = Convert.ToInt32(moment10 / areaPink);
                posY = Convert.ToInt32(moment01 / areaPink);
            }
            catch
            {
                //do nothing
            }
            Point center = new Point(posX, posY);
            MCvScalar colorCenter = new MCvScalar(0, 0, 255);
            MCvFont font = new MCvFont();
            MCvScalar colorFont = new MCvScalar(255, 0, 0);


            if (posX > 0 && posY > 0)
            {
                // draw circle around the center of the detected pink blob
                CvInvoke.cvCircle(image, center, 10, colorCenter, -1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                CvInvoke.cvInitFont(ref font,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX_SMALL, 1, 1, 0, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
                string textCenter = "(" + Convert.ToString(center.X) + "," +
Convert.ToString(center.Y) + ")";
                CvInvoke.cvPutText(image, textCenter, center, ref font, colorFont);
            }


            //Action message point and message color
            Point pointActionMessage = new Point(0, 30);
            MCvScalar colorMessage = new MCvScalar(0, 255, 255);
            string textStop = "Action:Stop";
            string textForwardRight = "Action:Forward Right";
            string textForwardLeft = "Action:Forward Left";
            string textForward = "Action:Forward";
            string textRotateLeft = "Action:Rotate Left";




            //searching and centralizing algorithm 6
            if (center.Y > 400) //440 originally
            {
                if (mission1Flag == 1)
                {
                    processingChoice = 11;  //again pink
                    mission1Flag = 0;
                    goHomeFlag = 1;
                }
                if (processingChoice == 8)   //play with the ball
                {
                    //give permission to rovio 2
                    processingChoice = 4;
                    playFlag = 1;
                }
                //stop Rovio 2
                rovio1.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);
```

```
            }
            else if (center.Y > 96)
                if (center.X > 512)
                {
                    rovio1.ManualDrive(8, speed);  //forward right
                    CvInvoke.cvPutText(image, textForwardRight, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 348)
                {

                    rovio1.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 256)
                {
                    rovio1.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 128)
                {
                    rovio1.ManualDrive(1, speed);  //forward
                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else
                {
                    rovio1.ManualDrive(7, speed);  //forward left
                    CvInvoke.cvPutText(image, textForwardLeft, pointActionMessage, ref font,
colorMessage);
                }
            else
            {
                rovio1.ManualDrive(17, speed); //rotate left
                CvInvoke.cvPutText(image, textRotateLeft, pointActionMessage, ref font,
colorMessage);
                rovio1.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);
            }



            return image;
        }

        /*******************************************/
        /*          Yellow Ball Commanding  Rovio 2   */
        /*******************************************/


        public Image<Bgr, Byte> YellowBallCommanding2(Image<Bgr, Byte> image)
        {

            MCvMoments moments = new MCvMoments();

            MCvScalar hsv_min = new MCvScalar(30, 101, 102);
```

```
            MCvScalar hsv_max = new MCvScalar(53, 174, 239);



            Image<Gray, Byte> thresholded = image.Convert<Gray, Byte>();

            Image<Hsv, Byte> hsv_image = image.Convert<Hsv, Byte>();

            CvInvoke.cvCvtColor(image, hsv_image,
Emgu.CV.CvEnum.COLOR_CONVERSION.CV_BGR2HSV);
            CvInvoke.cvInRangeS(hsv_image, hsv_min, hsv_max, thresholded);

            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvThreshold(thresholded, thresholded, 12, 256,
Emgu.CV.CvEnum.THRESH.CV_THRESH_BINARY);
            int iterations = 5;
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);


            CvInvoke.cvMoments(thresholded, ref moments, 1);

            double moment10 = CvInvoke.cvGetSpatialMoment(ref moments, 1, 0);
            double moment01 = CvInvoke.cvGetSpatialMoment(ref moments, 0, 1);
            double areaPink = CvInvoke.cvGetCentralMoment(ref moments, 0, 0);

            int posX = 0;
            int posY = 0;

            //exception handling
            try
            {
                posX = Convert.ToInt32(moment10 / areaPink);
                posY = Convert.ToInt32(moment01 / areaPink);
            }
            catch
            {
                //do nothing
            }
            Point center = new Point(posX, posY);
            MCvScalar colorCenter = new MCvScalar(0, 0, 255);
            MCvFont font = new MCvFont();
            MCvScalar colorFont = new MCvScalar(255, 0, 0);


            if (posX > 0 && posY > 0)
            {
                // draw circle around the center of the detected pink blob
                CvInvoke.cvCircle(image, center, 10, colorCenter, -1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                CvInvoke.cvInitFont(ref font,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX_SMALL, 1, 1, 0, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
                string textCenter = "(" + Convert.ToString(center.X) + "," +
Convert.ToString(center.Y) + ")";
                CvInvoke.cvPutText(image, textCenter, center, ref font, colorFont);
            }
```

```
            //Action message point and message color
            Point pointActionMessage = new Point(0, 30);
            MCvScalar colorMessage = new MCvScalar(0, 255, 255);
            string textStop = "Action:Stop";
            string textForwardRight = "Action:Forward Right";
            string textForwardLeft = "Action:Forward Left";
            string textForward = "Action:Forward";
            string textRotateLeft = "Action:Rotate Left";




            //searching and centralizing algorithm 6
            if (center.Y > 400) //440 originally
            {
                if (processingChoice == 8)   //play with the ball
                {
                    //give permission to rovio 2
                    processingChoice = 4;
                    playFlag = 1;
                }
                //stop Rovio 2
                rovio2.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);



            }
            else if (center.Y > 96)
                if (center.X > 512)
                {
                    rovio2.ManualDrive(8, speed);  //forward right
                    CvInvoke.cvPutText(image, textForwardRight, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 348)
                {

                    rovio2.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 256)
                {
                    rovio2.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 128)
                {
                    rovio2.ManualDrive(1, speed);  //forward
                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else
                {
                    rovio2.ManualDrive(7, speed);  //forward left
                    CvInvoke.cvPutText(image, textForwardLeft, pointActionMessage, ref font,
```

```
colorMessage);
                }
            else
            {
                rovio2.ManualDrive(17, speed); //rotate left
                CvInvoke.cvPutText(image, textRotateLeft, pointActionMessage, ref font,
colorMessage);
                rovio2.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);
            }



            return image;
        }



        /*******************************************/
        /*          New Ball Commanding  Rovio 1   */
        /*******************************************/


        public Image<Bgr, Byte> NewBallCommanding1(Image<Bgr, Byte> image)
        {

            MCvMoments moments = new MCvMoments();

            MCvScalar hsv_min = new MCvScalar(hue_min, sat_min, val_min);
            MCvScalar hsv_max = new MCvScalar(hue_max, sat_max, val_max);



            Image<Gray, Byte> thresholded = image.Convert<Gray, Byte>();

            Image<Hsv, Byte> hsv_image = image.Convert<Hsv, Byte>();

            CvInvoke.cvCvtColor(image, hsv_image,
Emgu.CV.CvEnum.COLOR_CONVERSION.CV_BGR2HSV);
            CvInvoke.cvInRangeS(hsv_image, hsv_min, hsv_max, thresholded);

            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvThreshold(thresholded, thresholded, 12, 256,
Emgu.CV.CvEnum.THRESH.CV_THRESH_BINARY);
            int iterations = 5;
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);



            CvInvoke.cvMoments(thresholded, ref moments, 1);

            double moment10 = CvInvoke.cvGetSpatialMoment(ref moments, 1, 0);
            double moment01 = CvInvoke.cvGetSpatialMoment(ref moments, 0, 1);
            double areaPink = CvInvoke.cvGetCentralMoment(ref moments, 0, 0);

            int posX = 0;
```

```
            int posY = 0;

            //exception handling
            try
            {
                posX = Convert.ToInt32(moment10 / areaPink);
                posY = Convert.ToInt32(moment01 / areaPink);
            }
            catch
            {
                //do nothing
            }
            Point center = new Point(posX, posY);
            MCvScalar colorCenter = new MCvScalar(0, 0, 255);
            MCvFont font = new MCvFont();
            MCvScalar colorFont = new MCvScalar(255, 0, 0);


            if (posX > 0 && posY > 0)
            {
                // draw circle around the center of the detected pink blob
                CvInvoke.cvCircle(image, center, 10, colorCenter, -1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                CvInvoke.cvInitFont(ref font,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_COMPLEX_SMALL, 1, 1, 0, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
                string textCenter = "(" + Convert.ToString(center.X) + "," +
Convert.ToString(center.Y) + ")";
                CvInvoke.cvPutText(image, textCenter, center, ref font, colorFont);
            }


            //Action message point and message color
            Point pointActionMessage = new Point(0, 30);
            MCvScalar colorMessage = new MCvScalar(0, 255, 255);
            string textStop = "Action:Stop";
            string textForwardRight = "Action:Forward Right";
            string textForwardLeft = "Action:Forward Left";
            string textForward = "Action:Forward";
            string textRotateLeft = "Action:Rotate Left";




            //searching and centralizing algorithm 6
            if (center.Y > 400) //440 originally
            {
                if (mission1Flag == 1)
                {
                    processingChoice = 11;  //again pink
                    mission1Flag = 0;
                    goHomeFlag = 1;
                }
                if (processingChoice == 8)   //play with the ball
                {
                    //give permission to rovio 2
                    processingChoice = 4;
                    playFlag = 1;
                }
                //stop Rovio 2
                rovio1.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);
```

```
            }
            else if (center.Y > 96)
                if (center.X > 512)
                {
                    rovio1.ManualDrive(8, speed);  //forward right
                    CvInvoke.cvPutText(image, textForwardRight, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 348)
                {

                    rovio1.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 256)
                {
                    rovio1.ManualDrive(1, speed);  //forward

                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else if (center.X > 128)
                {
                    rovio1.ManualDrive(1, speed);  //forward
                    CvInvoke.cvPutText(image, textForward, pointActionMessage, ref font,
colorMessage);
                }
                else
                {
                    rovio1.ManualDrive(7, speed);  //forward left
                    CvInvoke.cvPutText(image, textForwardLeft, pointActionMessage, ref font,
colorMessage);
                }
            else
            {
                rovio1.ManualDrive(17, speed); //rotate left
                CvInvoke.cvPutText(image, textRotateLeft, pointActionMessage, ref font,
colorMessage);
                rovio1.ManualDrive(0, speed); //stop
                CvInvoke.cvPutText(image, textStop, pointActionMessage, ref font,
colorMessage);
            }



            return image;
        }



                            /************************************/
                            /*          Color Tracking          */
                            /************************************/


        public Image<Gray, Byte> colorTracking(Image<Bgr, Byte> image,MCvScalar
hsv_min,MCvScalar hsv_max)
        {
            Image<Gray, Byte> thresholded = image.Convert<Gray, Byte>();
```

```
            Image<Hsv, Byte> hsv_image = image.Convert<Hsv, Byte>();

            CvInvoke.cvCvtColor(image, hsv_image,
Emgu.CV.CvEnum.COLOR_CONVERSION.CV_BGR2HSV);
            CvInvoke.cvInRangeS(hsv_image, hsv_min, hsv_max, thresholded);

            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvSmooth(thresholded, thresholded, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_BLUR,
9, 9, 0, 0);
            CvInvoke.cvThreshold(thresholded, thresholded, 12, 256,
Emgu.CV.CvEnum.THRESH.CV_THRESH_BINARY);
            int iterations = 5;
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvDilate(thresholded, thresholded, IntPtr.Zero, iterations);
            CvInvoke.cvErode(thresholded, thresholded, IntPtr.Zero, iterations);
            return thresholded;
        }

                            /*************************************************/
                            /*          Circle and color   detection         */
                            /*************************************************/

        public Image<Bgr, Byte> CircleAndColorDetection(Image<Bgr, Byte> image)
        {

            MCvScalar centreColor = new MCvScalar(0, 0, 0);  //black centre
            MCvScalar circumColorI = new MCvScalar(255, 255, 255);   //purple internal
circumfere
            MCvScalar circumColorE = new MCvScalar(255, 0, 0);   // white external circumf

            IntPtr cstorage = CvInvoke.cvCreateMemStorage(0);
            Image<Gray, Byte> gray = image.Convert<Gray, Byte>();
            Image<Gray, Byte> edge = image.Convert<Gray, Byte>();
            CvInvoke.cvThreshold(gray, gray, thresholdCircle, 256,
Emgu.CV.CvEnum.THRESH.CV_THRESH_BINARY);
            CvInvoke.cvSmooth(gray, gray, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_GAUSSIAN,param1,
param2,(double)param3,(double)param4);
            CvInvoke.cvCanny(gray, edge, (double)threshold1,
(double)threshold1,apertureSize);
            IntPtr circles = CvInvoke.cvHoughCircles(gray, cstorage,
Emgu.CV.CvEnum.HOUGH_TYPE.CV_HOUGH_GRADIENT,(double)accResolution,(double)minDist,(double)ca
nnyThreshold,(double)accThreshold,minRadius,maxRadius); //2, 100, 5, 50, 0, 100



            for (int i = 0; i < N; i++)   //filter N circles
            {
                unsafe
                {
                    try
                    {
                        float* p = (float*)CvInvoke.cvGetSeqElem(circles, i);
                        centre = new Point((int)p[0], (int)p[1]);

                        //CvInvoke.cvCircle(image, centre, 1, centreColor, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                        CvInvoke.cvCircle(image, centre, (int)p[2], circumColorI, 3,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                        CvInvoke.cvCircle(image, centre, (int)p[2] + 3, circumColorE, 3,
```

```
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED, 0);
                    string s = "Circle "+(i+1)+", Centre (" +
Convert.ToString(Convert.ToInt32(p[0])) + "," + Convert.ToString(Convert.ToInt32(p[1])) +
"), Radius: " + Convert.ToString(Convert.ToInt32(p[2]));
                    displayCirclePosition(s);
                }
                catch
                {
                    //do nothing
                }
            }

            //button clicked
            if (button20Clicked == true)    //take sample
            {

                Bgr bgr = image[centre];  //(0,0)
                centre.X += 10;
                Bgr bgr1 = image[centre]; //(10,0)
                centre.Y += 10;
                Bgr bgr2 = image[centre];  //(10,10)
                centre.X -= 10;
                Bgr bgr3 = image[centre];  //(0,10)
                centre.X -= 10;
                Bgr bgr4 = image[centre];  //(-10,10)
                centre.Y -= 10;
                Bgr bgr5 = image[centre];  //(-10,0)
                centre.Y -= 10;
                Bgr bgr6 = image[centre];  //(-10,-10)
                centre.X += 10;
                Bgr bgr7 = image[centre];  //(0,-10)
                centre.X += 10;
                Bgr bgr8 = image[centre];  //(10,-10)

                Bgr bgrAve = averageBGR(bgr, bgr1, bgr2, bgr3, bgr4, bgr5, bgr6, bgr7,
bgr8);


                Rgb rgb = new Rgb();
                rgb.Red = bgrAve.Red;
                rgb.Green = bgrAve.Green;
                rgb.Blue = bgrAve.Blue;
                string bgrString = "BGR Average: Centre : R[" + (int)rgb.Red + "] G[" +
(int)rgb.Green + "] B[" + (int)rgb.Blue + "]";
                displayColor(bgrString);
                hsv = RGB_to_HSV(rgb);
                string hsvString = "HSV: Centre : H[" + (int)hsv.Hue + "] S[" +
(int)hsv.Satuation + "] V[" + (int)hsv.Value + "]";
                displayColor(hsvString);

                button20Clicked = false;
            }

            //button clicked
            if (button45Clicked == true)
            {
                int hue_half_range=60;
                int sat_half_range=15;
                int val_half_range=15;

                hue_min = (int)hsv.Hue - hue_half_range;
                if (hue_min < 0)
                    hue_min = 0;
```

```
                sat_min = (int)hsv.Satuation - sat_half_range;
                if (sat_min < 0)
                    sat_min = 0;

                val_min = (int)hsv.Value - val_half_range;
                if (val_min < 0)
                    val_min = 0;



                hue_max = (int)hsv.Hue + hue_half_range;
                if (hue_max > 255)
                    hue_max = 255;

                sat_max = (int)hsv.Satuation + sat_half_range;
                if (sat_max > 255)
                    sat_max = 255;

                val_max = (int)hsv.Value + val_half_range;
                if (val_max > 255)
                    val_max = 255;


                button45Clicked = false;
            }
        }




            return image;
    }



    private void button23_Click(object sender, EventArgs e)  //edge detection button
    {

        if (processingChoice == 1)
        {
            processingChoice = 0;
        }
        else
        {
            trackBar1.Value = edparam1;
            label17.Text = "Thresh = " + Convert.ToInt32(trackBar1.Value);
            trackBar2.Value = edparam2;
            label18.Text = "ThreshLinking = " + Convert.ToInt32(trackBar2.Value);

            processingChoice = 1;
        }
    }

    private void button24_Click(object sender, EventArgs e)   //color detection
    {
        if (processingChoice == 2)
        {
            processingChoice = 0;
        }
        else
        {
```

```
                processingChoice = 2;
            }
        }

        private void button25_Click(object sender, EventArgs e)   //Normal Jpeg Display
button
        {

                processingChoice = 0;

        }

        private void button16_Click_1(object sender, EventArgs e)
        {
            if (processingChoice == 3)
            {
                processingChoice = 0;
            }
            else
            {
                trackBar3.Value = segments;
                label20.Text = "Segments = " + Convert.ToInt32(trackBar3.Value *
trackBar3.Value);

                processingChoice = 3;
            }
        }

        private void button18_Click(object sender, EventArgs e)  //rovio 2 find the
ball/commanding
        {
            if (processingChoice == 4)
            {
                processingChoice = 0;
            }
            else
            {
                processingChoice = 4;
            }


        }
        private void button41_Click(object sender, EventArgs e)
        {
            if (processingChoice == 6)
            {
                processingChoice = 0;
            }
            else
            {
                processingChoice = 6;
            }

        }

        private void button17_Click(object sender, EventArgs e)
        {
            if (processingChoice == 5)
            {
                processingChoice = 0;
            }
            else
            {
                processingChoice = 5;
```

```
        }
    }



     //pink ball commanding Rovio 1 & 2

    private void button43_Click(object sender, EventArgs e)
    {
        if (processingChoice == 8)
        {
            processingChoice = 0;
            playFlag = 0; //stop game flag
        }
        else
        {

            processingChoice = 8;
        }
    }

     //new ball commanding button rovio 1
    private void button21_Click(object sender, EventArgs e)
    {
        if (processingChoice == 12)
        {
            processingChoice = 0;
            playFlag = 0; //stop game flag
        }
        else
        {

            processingChoice = 12;
        }
    }

     //circle detection Rovio 1

    private void button44_Click(object sender, EventArgs e)
    {
        if (processingChoice == 7)
        {
            processingChoice = 0;
        }
        else
        {

            processingChoice = 7;

            numericUpDown1.Value = N;
            label11.Text = "threshold =" + Convert.ToString(thresholdCircle);
            numericUpDown2.Value = param1;
            numericUpDown3.Value = param2;
            numericUpDown4.Value = param3;
            numericUpDown5.Value = param4;
            numericUpDown6.Value = apertureSize;
            numericUpDown7.Value = threshold2;
            numericUpDown8.Value = threshold1;


            numericUpDown9.Value = accResolution;
            numericUpDown10.Value = minDist;
```

```
                numericUpDown11.Value = cannyThreshold;
                numericUpDown12.Value = accThreshold;
                numericUpDown13.Value = minRadius;
                numericUpDown14.Value = maxRadius;
            }
        }

        private void button49_Click(object sender, EventArgs e)  //find yellow ball rovio 1
        {
            if (processingChoice == 9)
            {
                processingChoice = 0;
            }
            else
            {

                processingChoice = 9;
            }
        }

        private void button50_Click(object sender, EventArgs e)  //find yellow ball rovio 2
        {
            if (processingChoice == 10)
            {
                processingChoice = 0;
            }
            else
            {

                processingChoice = 10;
            }
        }

        private void button51_Click(object sender, EventArgs e)   //mission 1
        {
            if (processingChoice == 11)
            {
                processingChoice = 0;
            }
            else
            {

                processingChoice = 11;
            }
        }


        /*******************************
         *                             *
         *     Method to diplay        *
         *     Execution Time in          *
         *     Gray Images             *
         *                             *
         *******************************/

        public Image<Gray, Byte> displayExecutionTimeInGray(Image<Gray, Byte>
grayImage,Stopwatch sw1)
        {
            //Frame delay display
            Point pointFrameDelayText = new Point(350, 20);
            MCvFont fontFrameDelayText = new MCvFont();
            CvInvoke.cvInitFont(ref fontFrameDelayText,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_PLAIN, 1, 1, 0, 1,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
```

```
            MCvScalar colorFrameDelayText = new MCvScalar(255, 255, 255);
            string strFrameDelay = "Execution Time:" + sw1.Elapsed.Milliseconds + "ms";
            CvInvoke.cvPutText(grayImage, strFrameDelay, pointFrameDelayText, ref
fontFrameDelayText, colorFrameDelayText);

            return grayImage;
        }


        /*******************************
         *                             *
         *     Method to diplay        *
         *     Execution Time in          *
         *     Gray Images             *
         *                             *
         *******************************/


        public Image<Bgr, Byte> displayExecutionTimeInColor(Image<Bgr, Byte> colorImage,
Stopwatch sw1)
        {
            //Frame delay display
            Point pointFrameDelayText = new Point(350, 20);
            MCvFont fontFrameDelayText = new MCvFont();
            CvInvoke.cvInitFont(ref fontFrameDelayText,
Emgu.CV.CvEnum.FONT.CV_FONT_HERSHEY_PLAIN, 1.3, 1.3, 0, 2,
Emgu.CV.CvEnum.LINE_TYPE.EIGHT_CONNECTED);
            MCvScalar colorFrameDelayText = new MCvScalar(0, 0, 255);
            string strFrameDelay = "Execution Time:" + sw1.Elapsed.Milliseconds + "ms";
            CvInvoke.cvPutText(colorImage, strFrameDelay, pointFrameDelayText, ref
fontFrameDelayText, colorFrameDelayText);
            return colorImage;
        }

        private void button46_Click(object sender, EventArgs e) //submit rovio 1
        {

            if (checkBox5.Checked)  //rovio 1
            {
                rovio1URL = "http://192.168.2.11";
            }
            else if(checkBox6.Checked)   //rovio 2
            {
                rovio1URL = "http://192.168.2.12";
            }
            else if(checkBox7.Checked)   //rovio 3
            {
                rovio1URL = "http://192.168.2.14";
            }
            else if(checkBox8.Checked)   //rovio 4
            {
                rovio1URL = "http://192.168.2.15";
            }
            else if (checkBox9.Checked)  //rovio 5
            {
                rovio1URL = "http://192.168.2.16";
            }
            /*   Create rovio object for rovio 1  AGAIN    */
            rovio1 = new RovioController("username", "password", rovio1URL);


        }

        private void button47_Click(object sender, EventArgs e) //submit rovio 2
```

```
        {
            if (checkBox5.Checked)  //rovio 1
            {
                rovio2URL = "http://192.168.2.11";
            }
            else if (checkBox6.Checked)   //rovio 2
            {
                rovio2URL = "http://192.168.2.12";
            }
            else if (checkBox7.Checked)   //rovio 3
            {
                rovio2URL = "http://192.168.2.14";
            }
            else if (checkBox8.Checked)   //rovio 4
            {
                rovio2URL = "http://192.168.2.15";
            }
            else if (checkBox9.Checked)  //rovio 5
            {
                rovio2URL = "http://192.168.2.16";
            }

            /*   Create rovio object for rovio 2  AGAIN   */
            rovio2 = new RovioController("username", "password", rovio2URL);


        }


         //display Rovios Coordinates from another thread - UI delegated calling

        public void displayPosRovio1(string coordinates)
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod theDelegateMethod = new
MyDelegateMethod(this.displayPosRovio1);
                this.Invoke(theDelegateMethod, new object[] { coordinates });
            }
            else
            {
                this.label8.Text = coordinates;
            }
        }


         //DisplayNameAttribute circle position in CircleF detection
        public void displayCirclePosition(string circlePosition)
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod7 theDelegateMethod = new
MyDelegateMethod7(this.displayCirclePosition);
                this.Invoke(theDelegateMethod, new object[] { circlePosition });
            }
            else
            {

                this.richTextBox1.AppendText(circlePosition+"\n");
                this.richTextBox1.ScrollToCaret();
            }
        }

        public void displayColor(string color)
```

```
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod8 theDelegateMethod = new
MyDelegateMethod8(this.displayColor);
                this.Invoke(theDelegateMethod, new object[] { color });
            }
            else
            {
                this.richTextBox2.AppendText(color + "\n");
                this.richTextBox2.ScrollToCaret();
            }
        }

        public void displayPosRovio2(string coordinates)
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod theDelegateMethod1 = new
MyDelegateMethod(this.displayPosRovio2);
                this.Invoke(theDelegateMethod1, new object[] { coordinates });
            }
            else
            {
                this.label9.Text = coordinates;
            }
        }

         //display distance between two rovios
        public void displayDistance(decimal distance)
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod5 theDelegateMethod = new
MyDelegateMethod5(this.displayDistance);
                this.Invoke(theDelegateMethod, new object[] { distance });
            }
            else
            {
                this.label10.Text = "Distance : "+Convert.ToString(distance);

            }
        }

        //bars for battery monitoring
        //roll progress bar 2

        public void rollProgressBar2()
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod2 theDelegateMethod2 = new
MyDelegateMethod2(this.rollProgressBar2);
                this.Invoke(theDelegateMethod2, new object[] {  });
            }
            else
            {

            if (this.progressBar2.Value < 100)
                this.progressBar2.Value += 10;
            else
                this.progressBar2.Value = 0;

            }
```

```
        }
        //roll progress bar 1
        public void rollProgressBar1()
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod1 theDelegateMethod3 = new
MyDelegateMethod1(this.rollProgressBar1);
                this.Invoke(theDelegateMethod3, new object[] { });
            }
            else
            {

                if (this.progressBar1.Value < 100)
                    this.progressBar1.Value += 10;
                else
                    this.progressBar1.Value = 0;

            }
        }

         //display battery for Rovio 1
        public void displayBar1(int battery)
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod3 theDelegateMethod4 = new
MyDelegateMethod3(this.displayBar1);
                this.Invoke(theDelegateMethod4, new object[] { battery });
            }
            else
            {
                if (battery >= 125)
                    this.progressBar1.Value = 100;
                else if (battery >= 123)
                    this.progressBar1.Value = 90;
                else if (battery >= 120)
                    this.progressBar1.Value = 80;
                else if (battery >= 117)
                    this.progressBar1.Value = 70;
                else if (battery >= 114)
                    this.progressBar1.Value = 60;
                else if (battery >= 111)
                    this.progressBar1.Value = 50;
                else if (battery >= 109)
                    this.progressBar1.Value = 40;
                else if (battery >= 107)
                    this.progressBar1.Value = 30;
                else
                    this.progressBar1.Value = 30;
                    //this.rovio1.GoHome();
            }
        }

        public void displayBar2(int battery2)
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod4 theDelegateMethod5 = new
MyDelegateMethod4(this.displayBar2);
                this.Invoke(theDelegateMethod5, new object[] { battery2 });
            }
            else
            {
```

```
                if (battery2 >= 125)
                    this.progressBar2.Value = 100;
                else if (battery2 >= 123)
                    this.progressBar2.Value = 90;
                else if (battery2 >= 120)
                    this.progressBar2.Value = 80;
                else if (battery2 >= 117)
                    this.progressBar2.Value = 70;
                else if (battery2 >= 114)
                    this.progressBar2.Value = 60;
                else if (battery2 >= 111)
                    this.progressBar2.Value = 50;
                else if (battery2 >= 109)
                    this.progressBar2.Value = 40;
                else if (battery2 >= 107)
                    this.progressBar2.Value = 30;
                else
                    this.progressBar2.Value = 30;
                    //this.rovio1.GoHome();
            }
        }

         //display ball position based on the positioning system
        public void displayBallPosition()
        {
            if (this.InvokeRequired)
            {
                MyDelegateMethod6 theDelegateMethod = new
MyDelegateMethod6(this.displayBallPosition);
                this.Invoke(theDelegateMethod, new object[] { });
            }
            else
            {
                //i deleted this label
                //this.label11.Text = "Ball Position: (" +
Convert.ToString(pos1x)+","+Convert.ToString(pos1y)+")" ;

            }
        }




         //truncate function for displaying decimal digits , theta and distance (2 decimal
digits)
        public decimal TruncateFunction(decimal number, int digits)
        {
            decimal stepper = (decimal)(Math.Pow(10.0, (double)digits));
            int temp = (int)(stepper * number);
            return (decimal)temp / stepper;
        }



        //Tab 2 manual controls for all the Rovios

        //rovio 1 manual

        private void button55_Click(object sender, EventArgs e) //forward
        {
            rovio1.ManualDrive(1, speed);
        }

        private void button59_Click(object sender, EventArgs e)  //down
```

```
        {
            rovio1.ManualDrive(2, speed);
        }

        private void button57_Click(object sender, EventArgs e)  //left
        {
            rovio1.ManualDrive(3, speed);
        }

        private void button52_Click(object sender, EventArgs e)  //right
        {
            rovio1.ManualDrive(4, speed);
        }

        private void button54_Click(object sender, EventArgs e)  //forward right
        {
            rovio1.ManualDrive(8, speed);
        }

        private void button56_Click(object sender, EventArgs e)  //forward left
        {
            rovio1.ManualDrive(7, speed);
        }

        private void button58_Click(object sender, EventArgs e) //back left
        {
            rovio1.ManualDrive(9, speed);
        }

        private void button53_Click(object sender, EventArgs e)   //back right
        {
            rovio1.ManualDrive(10, speed);
        }

        private void button60_Click(object sender, EventArgs e)    //stop
        {
            rovio1.ManualDrive(0, speed);
        }

        private void button98_Click(object sender, EventArgs e)    //rotate right
        {
            rovio1.ManualDrive(18, speed);
        }

        private void button97_Click(object sender, EventArgs e)    //rotate left
        {
            rovio1.ManualDrive(17, speed);
        }

        //rovio 2manual

        private void button75_Click(object sender, EventArgs e)  //forward
        {
            rovio2.ManualDrive(1, speed);
        }

        private void button71_Click(object sender, EventArgs e)  //back
        {
            rovio2.ManualDrive(2, speed);
        }

        private void button77_Click(object sender, EventArgs e)  //right
        {
            rovio2.ManualDrive(4, speed);
```

```
        }

        private void button73_Click(object sender, EventArgs e)  //left
        {
            rovio2.ManualDrive(3, speed);
        }

        private void button76_Click(object sender, EventArgs e)   //forward right
        {
            rovio2.ManualDrive(8, speed);
        }

        private void button74_Click(object sender, EventArgs e)   //forward left
        {
            rovio2.ManualDrive(7, speed);
        }

        private void button72_Click(object sender, EventArgs e)   //back left
        {
            rovio2.ManualDrive(9, speed);
        }

        private void button78_Click(object sender, EventArgs e)   //back right
        {
            rovio2.ManualDrive(10, speed);
        }

        private void button70_Click(object sender, EventArgs e)   //stop
        {
            rovio2.ManualDrive(0, speed);
        }

        private void button153_Click(object sender, EventArgs e)  //rotate right
        {
            rovio2.ManualDrive(18, speed);
        }

        private void button152_Click(object sender, EventArgs e) //rotate left
        {
            rovio2.ManualDrive(17, speed);
        }


         //rovio 3 manual
        private void button66_Click(object sender, EventArgs e)  //forawrd
        {
            rovio3.ManualDrive(1, speed);
        }

        private void button62_Click(object sender, EventArgs e)  //back
        {
            rovio3.ManualDrive(2, speed);
        }

        private void button64_Click(object sender, EventArgs e)  //left
        {
            rovio3.ManualDrive(3, speed);
        }

        private void button68_Click(object sender, EventArgs e)   //right
        {
            rovio3.ManualDrive(4, speed);
        }
```

```
        private void button67_Click(object sender, EventArgs e)   //forward right
        {
            rovio3.ManualDrive(8, speed);
        }

        private void button65_Click(object sender, EventArgs e)   //forwrad left
        {
            rovio3.ManualDrive(7, speed);
        }

        private void button63_Click(object sender, EventArgs e)   //back left
        {
            rovio3.ManualDrive(9, speed);
        }

        private void button69_Click(object sender, EventArgs e)   //back right
        {
            rovio3.ManualDrive(10, speed);
        }

        private void button147_Click(object sender, EventArgs e)   //rotate right
        {
            rovio3.ManualDrive(18, speed);
        }

        private void button146_Click(object sender, EventArgs e)  //rotate left
        {
            rovio3.ManualDrive(17, speed);
        }

        private void button61_Click(object sender, EventArgs e)  //stop
        {
            rovio3.ManualDrive(0, speed);
        }



         //rovio 4 manual

        private void button84_Click(object sender, EventArgs e)   //forward
        {
            rovio4.ManualDrive(1, speed);
        }

        private void button80_Click(object sender, EventArgs e)   //back
        {
            rovio4.ManualDrive(2, speed);
        }

        private void button82_Click(object sender, EventArgs e)   //left
        {
            rovio4.ManualDrive(3, speed);
        }

        private void button86_Click(object sender, EventArgs e)    //right
        {
            rovio4.ManualDrive(4, speed);
        }

        private void button85_Click(object sender, EventArgs e)   //forward right
        {
            rovio4.ManualDrive(8, speed);
        }
```

```
        private void button83_Click(object sender, EventArgs e)   //forward left
        {
            rovio4.ManualDrive(7, speed);
        }

        private void button81_Click(object sender, EventArgs e)   //back left
        {
            rovio4.ManualDrive(9, speed);
        }

        private void button87_Click(object sender, EventArgs e)   //back right
        {
            rovio4.ManualDrive(10, speed);
        }

        private void button151_Click(object sender, EventArgs e)   //rotate right
        {
            rovio4.ManualDrive(18, speed);
        }

        private void button150_Click(object sender, EventArgs e)   //rotate left
        {
            rovio4.ManualDrive(17, speed);
        }

        private void button79_Click(object sender, EventArgs e)   //stop
        {
            rovio4.ManualDrive(0, speed);
        }


         //rovio 5 manual

        private void button93_Click(object sender, EventArgs e)  //forawrd
        {
            rovio5.ManualDrive(1, speed);
        }

        private void button89_Click(object sender, EventArgs e)  //back
        {
            rovio5.ManualDrive(2, speed);
        }

        private void button91_Click(object sender, EventArgs e)  //left
        {
            rovio5.ManualDrive(3, speed);
        }

        private void button95_Click(object sender, EventArgs e)  //rihgt
        {
            rovio5.ManualDrive(4, speed);
        }

        private void button94_Click(object sender, EventArgs e)  //forward right
        {
            rovio5.ManualDrive(8, speed);
        }

        private void button92_Click(object sender, EventArgs e)  //forward left
        {
            rovio5.ManualDrive(7, speed);
        }

        private void button90_Click(object sender, EventArgs e)  //back left
```

```
        {
            rovio5.ManualDrive(9, speed);
        }

        private void button96_Click(object sender, EventArgs e)   //back right
        {
            rovio5.ManualDrive(10, speed);
        }

        private void button149_Click(object sender, EventArgs e)  //rotate right
        {
            rovio5.ManualDrive(18, speed);
        }

        private void button148_Click(object sender, EventArgs e)   //rotate left
        {
            rovio5.ManualDrive(17, speed);
        }

        private void button88_Click(object sender, EventArgs e)  //stop
        {
            rovio5.ManualDrive(0, speed);
        }

        private void button154_Click(object sender, EventArgs e) //enable threads for tab 2
        {
            enableTab2 = true;

            /*   Thread for taking images from Rovio 3      */
            Thread takeImagesThread3 = new Thread(new ThreadStart(takeImages3));
            takeImagesThread3.IsBackground = true;    //for the thread to close with the
application
            takeImagesThread3.Start();

            /*   Thread for taking images from Rovio 4      */
            Thread takeImagesThread4 = new Thread(new ThreadStart(takeImages4));
            takeImagesThread4.IsBackground = true;    //for the thread to close with the
application
            takeImagesThread4.Start();

            /*   Thread for taking images from Rovio 5      */
            Thread takeImagesThread5 = new Thread(new ThreadStart(takeImages5));
            takeImagesThread5.IsBackground = true;    //for the thread to close with the
application
            takeImagesThread5.Start();
        }

        private void button20_Click(object sender, EventArgs e)
        {
            button20Clicked = true;

        }

         //rgb to hsv method
        public static Hsv RGB_to_HSV(Rgb rgb)
        {
            int rgb_max = (int)Math.Max(rgb.Red, Math.Max(rgb.Green, rgb.Blue));
            int rgb_min = (int)Math.Min(rgb.Red, Math.Min(rgb.Green, rgb.Blue));
            Hsv hsv = new Hsv();
            hsv.Value = rgb_max;
            if (hsv.Value == 0)
            {
                hsv.Hue = hsv.Satuation = 0;
                return hsv;
```

```
            }
            hsv.Satuation = 255 * (rgb_max - rgb_min) / hsv.Value;
            if (hsv.Satuation == 0)
            {
                hsv.Hue = 0;
                return hsv;
            }
            /* Compute hue */
            if (rgb_max == rgb.Red)
            {
                hsv.Hue = 0 + 43 * (rgb.Green - rgb.Blue) / (rgb_max - rgb_min);
            }
            else if (rgb_max == rgb.Green)
            {
                hsv.Hue = 85 + 43 * (rgb.Blue - rgb.Red) / (rgb_max - rgb_min);
            }
            else /* rgb_max == rgb.b */
            {
                hsv.Hue = 171 + 43 * (rgb.Red - rgb.Green) / (rgb_max - rgb_min);
            }
            return hsv;

        }

        private void button45_Click(object sender, EventArgs e)
        {
            button45Clicked = true;

            trackBar4.Value = hue_min;
            trackBar5.Value = sat_min;
            trackBar6.Value = val_min;

            trackBar9.Value = hue_max;
            trackBar8.Value = sat_max;
            trackBar7.Value = val_max;

        }


        public static Bgr averageBGR(Bgr bgr, Bgr bgr1, Bgr bgr2, Bgr bgr3, Bgr bgr4, Bgr
bgr5, Bgr bgr6, Bgr bgr7, Bgr bgr8)
        {
            Bgr aveBgr = new Bgr();

            aveBgr.Blue = (bgr.Blue + bgr1.Blue + bgr2.Blue + bgr3.Blue + bgr4.Blue +
bgr5.Blue + bgr6.Blue + bgr7.Blue + bgr8.Blue) / 9;
            aveBgr.Green = (bgr.Green + bgr1.Green + bgr2.Green + bgr3.Green + bgr4.Green +
bgr5.Green + bgr6.Green + bgr7.Green + bgr8.Green) / 9;
            aveBgr.Red = (bgr.Red + bgr1.Red + bgr2.Red + bgr3.Red + bgr4.Red + bgr5.Red +
bgr6.Red + bgr7.Red + bgr8.Red) / 9;
            return aveBgr;
        }




    }
}
```

APPENDIX B – OPENCV METHODS

Edge Detection

```
#include "cv.h"
#include "highgui.h"
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main (int argc, char** argv )
{
        //exit key
       int c;
       //Initialise images that we will need
       IplImage *gray =0;
       IplImage *grayDown =0;
       IplImage *grayUp =0;
       //initialise capture structure for incoming video stream
       CvCapture* capture=cvCaptureFromCAM(0);
       //initialise the two windows that we will use for output
       cvNamedWindow("src",1);
       cvNamedWindow("grayDown",1);
       cvNamedWindow("grayUp",1);
       cvNamedWindow("gray",1);

       while(1)   //main infinite loop
       {
              //initialise the image which stores the incoming stream
              IplImage *src = 0;
              src=cvQueryFrame(capture);

              if( !src )    //if there is no frame exit the while(1)
              break;
```

```
        if( !gray )    //if there is no image gray, set the format of the secondary images
        {
           gray=cvCreateImage(cvGetSize(src),8,1);
           grayDown=cvCreateImage(cvSize( gray->width/2, gray->height/2 ),8,1);
           grayUp=cvCreateImage(cvGetSize(src),8,1);
        }

        //main image processing
        //convert to gray scale
        cvCvtColor(src,gray,CV_BGR2GRAY );
        //downscale the gray scale image
        cvPyrDown(gray,grayDown,CV_GAUSSIAN_5x5 );
        //upscale the downscaled image
        cvPyrUp(grayDown,grayUp,CV_GAUSSIAN_5x5 );
        //perform the canny edge detection algorithm
        //we can expirament with the up and down limit for
        //different results
        cvCanny(grayUp,gray,3,100,3 );

        //shoe the two windows
        cvShowImage("src",src);
        cvShowImage("grayDown",grayDown);
        cvShowImage("grayUp",grayUp);
        cvShowImage("gray",gray);

        //ready to exit loop
        c=cvWaitKey(10);
        if(c==27)break;
    }
    cvReleaseCapture( &capture);
    cvDestroyAllWindows();
}
```

Color Tracking

```
#include "cv.h"
#include "highgui.h"
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main (int argc, char** argv )
{
        //initialise key for exit and color choice
        int c,key;
        //number of iteration for dilationa and erosion
        int iterations = 5;
        //initialization of the color range ,here for blue
        CvScalar hsv_min={110,190,0};
        CvScalar hsv_max={120, 225, 255};

      //initialise the two secondary images that we are going to use
      IplImage *inRange =0;
       IplImage *thresholded =0;
       IplImage *cthresholded =0;
       IplImage *hsv =0;
       IplImage *smoothed =0;

       //capture struct for video stream
       CvCapture* capture=cvCaptureFromCAM(0);

       //initialise the two windows
       cvNamedWindow("src",1);
       cvNamedWindow("hsv",1);
       cvNamedWindow("inRange",1);
       cvNamedWindow("thresholded",1);
```

```
	//here is the menu that corresponds to the tick
	//menu in my Visual C# application
	//for the Rovios

	//printf("      Menu    \n");
	//choose color detection
	//printf("Choose Color    \n");
//	  printf("b: BLUE        \n");
//	  printf("p: PURPLE      \n");
//	  printf("y: YELLOW      \n");
//	  printf("n: Nothing     \n");
//	  printf("Enter Character    \n");
//	  scanf("%d\n",&key);
//	  printf("you chose:%d\n",key);
//	  if (key==1)
//		{
//		   CvScalar hsv_min={110,50,110};
//		   CvScalar hsv_max={124, 180, 200};
//		}
//	  else if (key==2)
//		{
//		    CvScalar hsv_min={125, 50, 110};
//		   CvScalar hsv_max={150, 180, 200};
//		}
//	  else if (key==3)
//		{
//		    CvScalar hsv_min={31, 50, 180};
//		   CvScalar hsv_max={40, 200, 200};
//		}


	while(1) //main loop
	{
```

```
IplImage *src = 0;
src=cvQueryFrame(capture);

if( !src )    //if there is no frame exit the while(1)
break;

if( !thresholded )    //if there is no image thresholded, do the followings
{
   hsv=cvCreateImage(cvGetSize(src),8,3);
   thresholded=cvCreateImage(cvGetSize(src),8,1);
   cthresholded=cvCreateImage(cvGetSize(src),8,1);
   inRange=cvCreateImage(cvGetSize(src),8,1);
   smoothed=cvCreateImage(cvGetSize(src),8,1);
}
//convert to HSV
cvCvtColor(src, hsv, CV_BGR2HSV);
//search for the given range
cvInRangeS(hsv, hsv_min, hsv_max, inRange);

//filtering the gray image
cvSmooth(inRange, smoothed, CV_BLUR, 9, 9, 0, 0);
//cvSmooth(smoothed, smoothed, CV_BLUR, 9, 9, 0, 0);
//cvSmooth(smoothed, smoothed, CV_BLUR, 9, 9, 0, 0);

//detecting the appropriate range in the gray scale
cvThreshold(smoothed, thresholded, 12, 256, CV_THRESH_BINARY);

//opening and closing
cvErode(thresholded, thresholded, NULL, iterations);
cvDilate(thresholded, thresholded, NULL, iterations);
cvDilate(thresholded, thresholded, NULL, iterations);
cvErode(thresholded, thresholded, NULL, iterations);
```

```
            //show the two images
            cvShowImage("src",src);
            cvShowImage("hsv",hsv);
            cvShowImage("inRange",inRange);
            cvShowImage("smoothed",smoothed);
            cvShowImage("thresholded",thresholded);

            //ready to exit loop
            c=cvWaitKey(10);
            if(c==27)break;
        }
        cvReleaseCapture( &capture);
        cvDestroyAllWindows();
}
```

Image Segmentation

```
#include "cv.h"
#include "highgui.h"
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main (int argc, char** argv )
{
        //key for exit
        int c;
        //for the number of segments on the image (nxn= the total segments)
        //expirement with this variable for different results
        int n=8;
        //counters for loop which creates the points
        int i,j;
        //color of the rectangles
        CvScalar colorRect={0,255,255};
```

```
//text color
CvScalar colorText={0,0,255};
//initialise the font that we re goin to use
CvFont font;
//initialise the strings for the display of the two points of each rectangle
char string_p1[40];
char string_p2[40];
//initialisation of segmented image
IplImage *segment =0;
//struct capture
CvCapture* capture=cvCaptureFromCAM(0);
//initialise the two windows
cvNamedWindow("src",1);
cvNamedWindow("segment",1);

while(1)  //infinite loop
{
      IplImage *src = 0;
      src=cvQueryFrame(capture);

      if( !src )    //if there is no frame exit the while(1)
      break;

      if( !segment )    //if there is no image segment, set its format
      {
         segment=cvCreateImage(cvGetSize(src),8,3);
      }
      //copy the src to image segment
      cvCopy(src,segment,0);

      //two for loops
      //for creating the various points
      //and display them on the image segment
```

```
//n X n : the total number of segments
for (i = 0; i <= n; i++)
{
    for (j = 0; j <= n; j++)
    {
       //create the points
       CvPoint p1={i * src->width / n, j*src->height / n};
       CvPoint p2={(i + 1) * src->width / n, (j+1)*src->height / n};

       //initialise the font that we'll use
       cvInitFont(&font,CV_FONT_HERSHEY_COMPLEX,0.5,0.5, 0,1, 8);

       //create the string
       //which will display point 1
       //of each rectangle
       //begin the string with the "("
       strcpy( string_p1, "(");
       char p1_x[5];
       //convert int to string
       itoa(p1.x, p1_x, 10);
       //copy to the end of the previous string
       strcat( string_p1, p1_x);
       //add comma
       strcat( string_p1,",");
       char p1_y[5];
       //convert int to string
       itoa(p1.y, p1_y, 10);
       //add this to the current end
       strcat( string_p1, p1_y);
       //complete the string for display
       strcat( string_p1,")");
       //display the final string
       cvPutText(segment, string_p1, p1, &font, colorText);
       //display the rectangles based on the two points
```

```
                    cvRectangle(segment, p1, p2, colorRect, 1, 8, 0);
                }
            }
            //show the two images
            cvShowImage("src",src);
            cvShowImage("segment",segment);

            //ready to exit loop
            c=cvWaitKey(10);
            if(c==27)break;
        }
        cvReleaseCapture( &capture);
        cvDestroyAllWindows();
}
```

Pink Ball Tracking

```
#include "cv.h"
#include "highgui.h"
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>

int main (int argc, char** argv )
{
        //initialise the string to display the centre of the detected area
        char textCenter[40];
        //exit key
        int c;
        //iterations for erosion and dilation
        int iterations = 5;
        //initialise secondary images
        IplImage *thresholded =0;
        IplImage *thresholded2 =0;
```

```
IplImage *hsv =0;
//moments to calculate the centre
CvMoments moments;
//struct of the centre
CvPoint centre;
//initialise the color for the centre and the font in RGB
CvScalar colorCenter={0, 255, 255};
CvScalar colorFont={255, 0, 0};
//initialise the font
CvFont font;
//the twopink color ranges for the detection of the pink ball
CvScalar hsv_min={0, 50, 170};
CvScalar hsv_max={10, 180, 256};
CvScalar hsv_min2={170, 50, 170};
CvScalar hsv_max2={256, 180, 26};
//structure for capturing the images
CvCapture* capture=cvCaptureFromCAM(0);
//create two windows to diplay the output
cvNamedWindow("src",1);
cvNamedWindow("thresholded",1);

while(1)   //infinite main loop
{
      IplImage *src = 0;
      src=cvQueryFrame(capture);

      if( !src )    //if there is no frame exit the while(1)
      break;

      if( !thresholded )    //if there is no image thresholded, do the followings
      {

         thresholded=cvCreateImage(cvGetSize(src),8,1);
         thresholded2=cvCreateImage(cvGetSize(src),8,1);
```

```
    hsv=cvCreateImage(cvGetSize(src),8,3);


}
//convert image to HSV format
cvCvtColor(src, hsv, CV_BGR2HSV);
//search in the first range of pink in HSV
cvInRangeS(hsv, hsv_min, hsv_max, thresholded);
//search in the second range of pink color in HSV
cvInRangeS(hsv, hsv_min2, hsv_max2, thresholded2);
//logical or between the two gray thresholded images
cvOr(thresholded, thresholded2, thresholded, 0);
//filter - smoothing - BLUR
cvSmooth(thresholded, thresholded, CV_BLUR, 9, 9, 0, 0);
cvSmooth(thresholded, thresholded, CV_BLUR, 9, 9, 0, 0);
cvSmooth(thresholded, thresholded, CV_BLUR, 9, 9, 0, 0);
//threshold to hold the disired area
cvThreshold(thresholded, thresholded, 12, 256, CV_THRESH_BINARY);
//opening and closing - filter the small white areas
cvErode(thresholded, thresholded, 0, iterations);
cvDilate(thresholded, thresholded, 0, iterations);
cvDilate(thresholded, thresholded, 0, iterations);
cvErode(thresholded, thresholded, 0, iterations);
//hold moments from the final gray image
cvMoments(thresholded, &moments, 1);
//calculate the spatial and central moments
double moment10 = cvGetSpatialMoment(&moments, 1, 0);
double moment01 = cvGetSpatialMoment(&moments, 0, 1);
double areaPink = cvGetCentralMoment(&moments, 0, 0);
//initialise the position x & y
int posX = 0;
int posY = 0;
//calculate position x & y
posX = cvRound(moment10 / areaPink);
posY = cvRound(moment01 / areaPink);
```

```
            //give to the struct centre
            centre.x=posX;
            centre.y=posY;
            //if centre position positive (acceptable values)
            if (posX > 0 && posY > 0)
            {
           // draw circle and display the position of the gravity center of the pink blob detected
           cvCircle(src, centre, 10, colorCenter, -1, 8, 0);
           cvInitFont(&font, CV_FONT_HERSHEY_COMPLEX_SMALL, 1, 1, 0, 1, 8);
           strcpy( textCenter, "(");
           char p1_x[5];
           itoa(posX, p1_x, 10);
           strcat( textCenter, p1_x);
           strcat( textCenter,",");
           char p1_y[5];
           itoa(posY, p1_y, 10);
           strcat( textCenter, p1_y);
           strcat( textCenter,")");
           cvPutText(src, textCenter, centre, &font, colorFont);
           }
            //show the two windows as an output
            cvShowImage("src",src);
            cvShowImage("thresholded",thresholded);



            //ready to exit loop
            c=cvWaitKey(10);
            if(c==27)break;
      }
      cvReleaseCapture( &capture);
      cvDestroyAllWindows();
}
```

Circle Detection and Dynamic Color Range Adjustment

```
#include "cv.h"
#include "highgui.h"
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>



#define MIN3(x,y,z)  ((y) <= (z) ? ((x) <= (y) ? (x) : (y)) : ((x) <= (z) ? (x) : (z)))
#define MAX3(x,y,z)  ((y) >= (z) ? ((x) >= (y) ? (x) : (y)) : ((x) >= (z) ? (x) : (z)))

struct rgb_color
{
   unsigned char r, g, b;    /* Channel intensities between 0 and 255 */
};

struct hsv_color
{
   unsigned char hue;       /* Hue degree between 0 and 255 */
   unsigned char sat;       /* Saturation between 0 (gray) and 255 */
   unsigned char val;       /* Value between 0 (black) and 255 */
};

struct hsv_color rgb_to_hsv(struct rgb_color rgb)
{
   struct hsv_color hsv;
   unsigned char rgb_min, rgb_max;
   rgb_min = MIN3(rgb.r, rgb.g, rgb.b);
   rgb_max = MAX3(rgb.r, rgb.g, rgb.b);
   hsv.val = rgb_max;
   if (hsv.val == 0)
   {
   hsv.hue = hsv.sat = 0;
```

```
    return hsv;
    }
    hsv.sat = 255*(rgb_max - rgb_min)/hsv.val;
    if (hsv.sat == 0)
    {
    hsv.hue = 0;
    return hsv;
    }
    /* Compute hue */
    if (rgb_max == rgb.r)
    {
    hsv.hue = 0 + 43*(rgb.g - rgb.b)/(rgb_max - rgb_min);
    }
    else if (rgb_max == rgb.g)
    {
    hsv.hue = 85 + 43*(rgb.b - rgb.r)/(rgb_max - rgb_min);
    }
    else /* rgb_max == rgb.b */
    {
    hsv.hue = 171 + 43*(rgb.r - rgb.g)/(rgb_max - rgb_min);
    }
    return hsv;
}



int main (int argc, char** argv )
{
        //initialise structures
       struct rgb_color rgb;
       struct hsv_color hsv;

        int thresh = 50;
        CvMemStorage* fstorage = 0;
```

```
CvScalar centreColor={0,0,0};   //black centre
CvScalar circumColorI={255,255,255};   //purple centre Internal
CvScalar circumColorE={255,0,0};   //purple centre External

//averaging the values of the colors
int row_x=0,row_x1=0,row_x2=0;
int total=0;

int c,x,y,key;
int px[0], py[0];
int edge_thresh = 1;
IplImage *csrc=0;
IplImage *gray =0;
IplImage *edge =0;
IplImage *thresholded =0;
IplImage *hsv2 =0;

CvMemStorage* cstorage = cvCreateMemStorage(0);
fstorage= cvCreateMemStorage(0);

//get the video from webcam
CvCapture* capture=cvCaptureFromCAM(0);

//initialese windows
cvNamedWindow("src",1);
cvNamedWindow("gray",1);
//cvNamedWindow("hsv",1);

while(1)
{
      IplImage *src = 0;
      src=cvQueryFrame(capture);
      if( !src )    //if there is no frame exit the while(1)
      break;
```

```
        if( !csrc )    //if there is no image csrc, do the followings
        {
           csrc=cvCreateImage(cvGetSize(src),8,3);
           gray=cvCreateImage(cvGetSize(src),8,1);
           edge=cvCreateImage(cvGetSize(src),8,1);
        }
        csrc=cvCloneImage(src);
        //cvCvtColor(src,hsv,CV_BGR2HSV);
        cvCvtColor(src,gray,CV_BGR2GRAY);
        gray->origin=1;
        cvThreshold(gray,gray,100,255,CV_THRESH_BINARY);
        cvSmooth( gray, gray, CV_GAUSSIAN, 11, 11 ,0,0);
        cvCanny(gray, edge, (float)edge_thresh*3, (float)edge_thresh*7, 5); //cvCanny(gray, edge, (float)edge_thresh, (float)edge_thresh*3, 5);
        CvSeq* circles =  cvHoughCircles( gray, cstorage, CV_HOUGH_GRADIENT, 2, gray->height/10, 5,50, 0,100 );   //5,35,0,0

        int i;
        for( i = 0; circles->total>=1?i<1:i < circles->total; i++ ) //just make a filter to limit only <=2 circles to draw
        {
            float* p = (float*)cvGetSeqElem( circles, i );
            cvCircle( src, cvPoint(cvRound(p[0]),cvRound(p[1])), 1, centreColor, 1, 8, 0 ); //black circle centre (dot)
            cvCircle( src, cvPoint(cvRound(p[0]),cvRound(p[1])), cvRound(p[2])-3, circumColorI, 3, 8, 0 );  //circle circuference
            cvCircle( src, cvPoint(cvRound(p[0]),cvRound(p[1])), cvRound(p[2]), circumColorE, 3, 8, 0 );  //circle circuference
            px[i]=cvRound(p[0]);
            py[i]=cvRound(p[1]);
            printf("Centre (%d,%d), Radius: %d\n",px[0],py[0],cvRound(p[2]));
        }
```

```
            key=cvWaitKey(10);

            if (key=='a')  //sampling
            {
             printf("\n\n"); //two new lines

             for (y=0;y<src->height;y++)
             {
               unsigned char* row=&CV_IMAGE_ELEM(src,unsigned char,y,0);
               for(x=0;x<src->width*src->nChannels;x+=src->nChannels)
               {
                     //search for the black centre still in RGB? Why?
                     if( row[x]==0 && row[x+1]==0 && row[x+2]==0)
                     {
                        total++;
                        row_x+=row[x+12];
                        row_x1+=row[x+13];
                        row_x2+=row[x+14];

                        printf("Centre : R[%d] G[%d] B[%d] \n",row[x+14],row[x+13],row[x+12]);
                     }
               }

             }
             printf("\n Average : R[%d] G[%d] B[%d] \n",cvRound(row_x2/total),cvRound(row_x1/total),cvRound(row_x/total));
             rgb.r=cvRound(row_x2/total);
             rgb.g=cvRound(row_x1/total);
             rgb.b=cvRound(row_x/total);
             hsv = rgb_to_hsv(rgb);
             printf("\n >>Average : H[%d] S[%d] V[%d] <<\n\n",hsv.hue, hsv.sat, hsv.val);
             printf("\tMenu\n");
             printf(" a - Sample Color Again\n");
             printf(" c - Find color \n\n");
```

```
}


key=cvWaitKey(10);

if(key=='c')
{
	//set half ranges of the HSV format
	int hue_half_range=90;
	int sat_half_range=15;
	int val_half_range=15;

	//set hsv range
	int hsv_min_hue=hsv.hue-hue_half_range;
	if(hsv_min_hue<0)
		hsv_min_hue=0;

	int hsv_min_sat=hsv.sat-sat_half_range;
	if(hsv_min_sat<0)
		hsv_min_sat=0;

	int hsv_min_val=hsv.val-val_half_range;
	if(hsv_min_val<0)
		hsv_min_val=0;

	int hsv_max_hue=hsv.hue+hue_half_range;
	if(hsv_max_hue>255)
		hsv_max_hue=255;

	int hsv_max_sat=hsv.sat+sat_half_range;
	if(hsv_max_sat>255)
		hsv_max_sat=255;

	int hsv_max_val=hsv.val+val_half_range;
```

```
                    if(hsv_max_val>255)
                        hsv_max_val=255;

                    CvScalar hsv_min={hsv_min_hue, hsv_min_sat, hsv_min_val};  //hsv_min_val
                    CvScalar hsv_max={hsv_max_hue, hsv_max_sat, hsv_max_val};
//hsv_max_val

                    //print hsv range
                    printf("\n HSV_MIN : H[%d] S[%d] V[%d] \n",hsv_min_hue, hsv_min_sat,
hsv_min_val);
                    printf("\n HSV_MAX : H[%d] S[%d] V[%d] \n",hsv_max_hue, hsv_max_sat,
hsv_max_val);
                    //iterations
                    int iterations=5;
                    CvMoments moments;
                    CvPoint centre;
                    //initialise the color for the centre and the font in RGB
                    CvScalar colorCenter={255, 255, 0};
                    CvScalar colorFont={255, 0, 0};
                    //initialise the font
                    CvFont font;
                     //initialise the string to display the centre of the detected area
                     char textCenter[40];

                   while(1) //infinite loop
                   {
                        IplImage *src = 0;
                        src=cvQueryFrame(capture);

                        if( !src )    //if there is no frame exit the while(1)
                        break;

                        if( !thresholded )    //if there is no image thresholded, do the followings
                        {
```

```
  thresholded=cvCreateImage(cvGetSize(src),8,1);
  hsv2=cvCreateImage(cvGetSize(src),8,3);

}
//convert image to HSV format
cvCvtColor(src, hsv2, CV_BGR2HSV);
//search in the first range of pink in HSV
cvInRangeS(hsv2, hsv_min, hsv_max, thresholded);
//filter - smoothing - BLUR
cvSmooth(thresholded, thresholded, CV_BLUR, 9, 9, 0, 0);
//cvSmooth(thresholded, thresholded, CV_BLUR, 9, 9, 0, 0);
//cvSmooth(thresholded, thresholded, CV_BLUR, 9, 9, 0, 0);
//threshold to hold the disired area
cvThreshold(thresholded, thresholded, 12, 255, CV_THRESH_BINARY);
//opening and closing - filter the small white areas
cvErode(thresholded, thresholded, 0, iterations);
cvDilate(thresholded, thresholded, 0, iterations);
cvDilate(thresholded, thresholded, 0, iterations);
cvErode(thresholded, thresholded, 0, iterations);
//hold moments from the final gray image
cvMoments(thresholded, &moments, 1);
//calculate the spatial and central moments
double moment10 = cvGetSpatialMoment(&moments, 1, 0);
double moment01 = cvGetSpatialMoment(&moments, 0, 1);
double areaPink = cvGetCentralMoment(&moments, 0, 0);
//initialise the position x & y
int posX = 0;
int posY = 0;
//calculate position x & y
posX = cvRound(moment10 / areaPink);
posY = cvRound(moment01 / areaPink);
//give to the struct centre
centre.x=posX;
```

```
                    centre.y=posY;
                    //if centre position positive (acceptable values)
                    if (posX > 0 && posY > 0)
                    {
                   // draw circle and display the position of the gravity center of the pink blob
detected
                    cvCircle(src, centre, 10, colorCenter, -1, 8, 0);
                    cvInitFont(&font, CV_FONT_HERSHEY_COMPLEX_SMALL, 1, 1, 0, 1,
8);
                    strcpy( textCenter, "(");
                    char p1_x[5];
                    itoa(posX, p1_x, 10);
                    strcat( textCenter, p1_x);
                    strcat( textCenter,",");
                    char p1_y[5];
                    itoa(posY, p1_y, 10);
                    strcat( textCenter, p1_y);
                    strcat( textCenter,")");
                    cvPutText(src, textCenter, centre, &font, colorFont);
                    }
                    //show the two windows as an output
                    cvShowImage("src",src);
                    cvShowImage("thresholded",thresholded);


                    //ready to exit loop
                  c=cvWaitKey(10);
                  if(c==27)break;
                }
            }


        cvShowImage("src",src);
        cvShowImage("gray",gray);
```

```
            //cvShowImage("hsv",hsv);
            //release memory
            cvReleaseImage(&csrc);
            cvClearMemStorage( fstorage );
            //ready to exit loop
            c=cvWaitKey(10);
            if(c==27)break;
        }
        cvReleaseCapture( &capture);
        cvDestroyAllWindows();
}
```