UiT THE ARCTIC UNIVERSITY OF NORWAY

Faculty of Science and Technology

Department of Computer Science

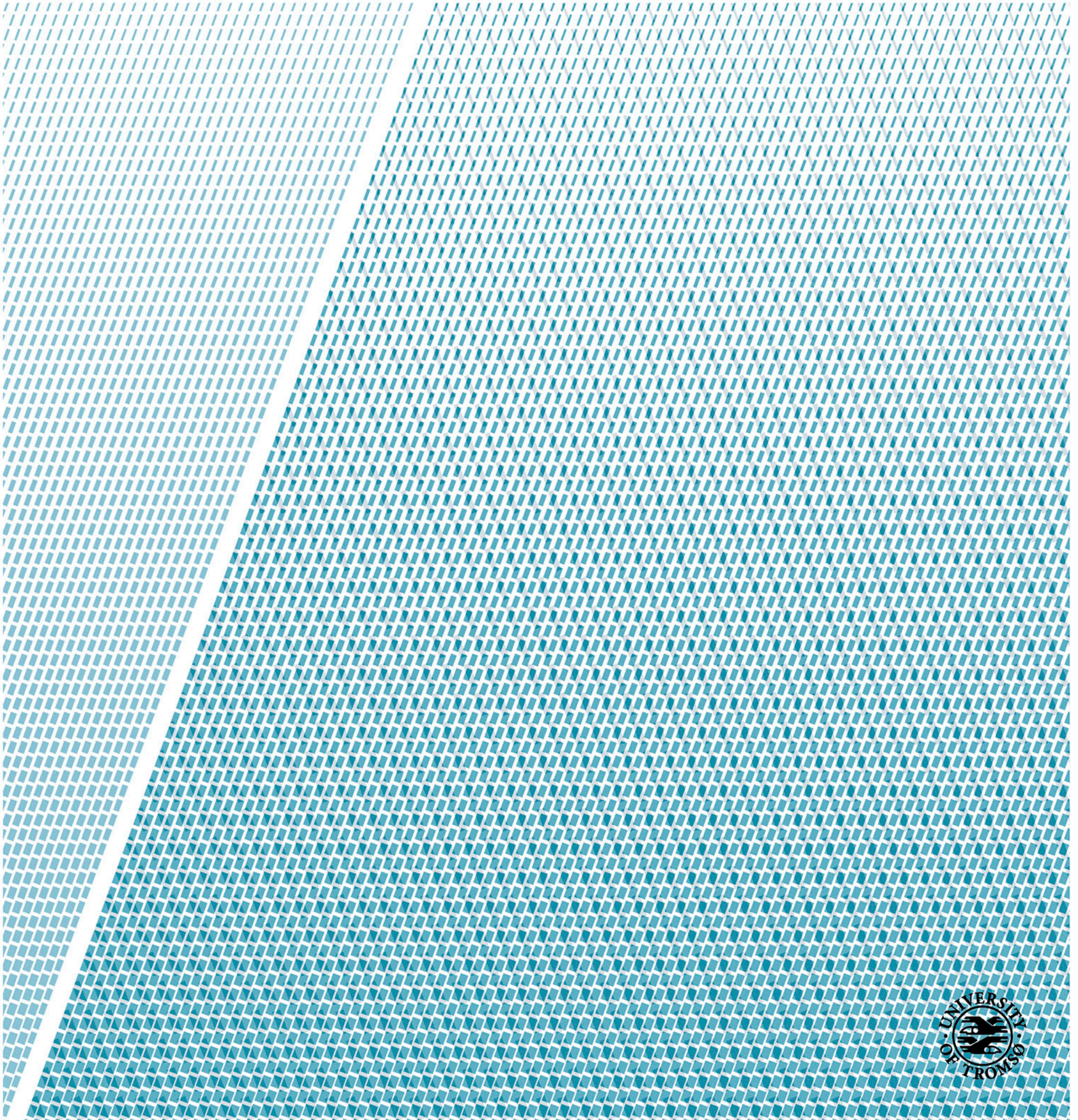
Sphero NAV

A Robotic Navigation and Control Platform

—

Simon Andreas Engstrøm Nistad

INF-3981 Master's Thesis in Computer Science, June 2014



I. Abstract

The computer science department at the University of Tromso has in the later years invested in different robotic platforms for use in their recruitment program. The recruitment program is responsible for targeting high school students and open their eyes for the world of computer science. The use of robots in recruitment is an approach to give a simple demonstration of what simple programming skills can achieve.

One of the limitations addressed by the administrators of the recruitment program is the robotic devices lacking possibilities in navigation and positioning. The devices used by the university are missing sensors that can obtain data about their accurate position. The administrators has posted request for a platform that would allow for position tracking of the devices. The system would make it possible to create richer and attention-grabbing applications for recruitment events and school visits.

This thesis addresses this request and presents Sphero NAV, a camera based navigation and control platform for the robotic ball Orbotix Sphero. Sphero NAV is a Python library that serves as a base for developing new Sphero applications. The library allows for communication and control of one or multiple Sphero devices using Sphero's rich API and functionality. Sphero NAV implements a tracker system that uses image based position tracking. A camera mounted in the celling over the tracking area captures a video stream. The system analyzes the images and locates the devices position.

The evaluation of Sphero NAV shows that the library implements is a simple but efficient image-based position and control system that developers can use to create different applications for recruitment purposes.

II. Acknowledgements

This thesis is dedicated to the memory of a beloved family member, Harry Jardine, who passed away last Christmas.

I would like to thank my advisor, John Markus Bjørndalen for providing guidance and feedback throughout this project.

I would also like to thank my fellow students and good friends through five years of study: Alexander Svendsen, Ida Jaklin Johansen, Simen Lomås Johannessen, Steffen Hageland and Tom Pedersen for good discussions, support and many good memories during my entire study. I look forward to work with some of you as colleagues after the summer.

A special thanks goes to Alexander that I have shared office with during the time of this thesis. Thank you for feedback and good discussions during this project.

A thank goes to my close family and friends who always supports me and believes in my decisions.

Finally, special thanks go to Karina Byrkjeland who always knows how to cheer me up. Thank you for your support and love.

Table of content

I. Abstract I
II. Acknowledgements III
III. List of figures IX
IV. List of tables XI
Chapter 1 - Introduction 1
1.1 Overview 1
1.2 Problem definition 3
1.3 Envisioned system 3
1.4 Contributions 5
1.5 Limitations 5
1.6 Outline 5
Chapter 2 - Related work 7
2.1 Introduction 7
2.2 Bouncing Star 7
2.3 Ping pong plus 8
2.4 Open Pool 9
2.5 Pixelbots (Display swarm) 10
Chapter 3 - Orbotix Sphero 13
3.1 Introduction 13
3.2 The device 14
3.2.1 SDK's 15

3.3 System Design 16
3.3.1 Client – Server 16
3.3.2 Virtual devices 16
3.4 Sphero Overview 17
3.4.1 RGB light 18
3.4.2 Coordinate system 18
3.4.3 Locator 19
3.4.4 OrbBasic 20
3.4.5 Macros 20
3.5 Packet structure 20
3.5.1 Synchronous packets 20
3.5.2 Synchronous responses 22
3.5.3 Asynchronous packets 23
Chapter 4 - Sphero NAV 25
4.1 Introduction 25
4.2 Architecture 26
4.3 Design 29
4.3.1 Tracker 29
4.3.2 Sphero Module 33
4.3.3 PS3 Module 35
4.4 Use Case 36
4.4.1 Application ideas 36
4.4.2 API usage examples 36
Chapter 5 - Implementation 43
5.1 Introduction 43
5.2 Technologies used 43
5.3 Object Tracking 44
5.3.1 Algorithm 44

5.3.2 Traceable object and sample class 47
5.3.3 Filter 48
5.3.4 Camera controller 48
5.4 Sphero Module 49
5.4.1 Communication 49
5.4.2 Sphero Streaming 51
5.4.3 Sphero Manager 52
5.4.4 Sphero Calibration 53
5.5 PS3 Module 55
Chapter 6 - Evaluation 57
6.1 Introduction 57
6.2 Experiments 57
6.2.1 The experimental environment 57
6.2.2 Communication 57
6.2.3 Tracking performance 60
6.2.4 Library test 62
6.2.5 Video 63
6.3 Known bugs and issues 64
6.3.1 Spikes in communication 64
6.3.2 Microsoft Kinect 65
6.3.3 Color tracking 65
6.3.4 Internal reference heading 65
6.4 Improvements 66
6.4.1 Support all platforms 66
6.4.2 Bluetooth lookup is slow 66
6.4.3 Distributed system 66
6.4.4 Image evaluation 67
6.4.5 Improved Tracking 68

6.5 Problem definition solved 70
Chapter 7 - Conclusion 73
7.1 Conclusion 73
7.2 Concluding remarks 74
7.3 Future work and ideas 74
Chapter 8 - References 77

III. List of figures

Figure 1 - Conceptual architecture 4
Figure 2 - Shows the Sphero device 13
Figure 3 - Shows the inside and IMU of the Sphero 14
Figure 4 - Sphero client --> server model..16
Figure 5 - Accelerometer and Gyroscope in Sphero 17
Figure 6 - Sphero angles vs. Euclidean 19
Figure 7 - Sphero NAV logo 25
Figure 8 - Architecture of Sphero NAV...26
Figure 9 - Manager Design pattern used for Sphero and PS3 module 28
Figure 10 – Tracker design 29
Figure 11 – Tracking mask displayed by the tracker 30
Figure 12 - GUI of the camera settings manager 31
Figure 13 - Tracker GUI 32
Figure 14 – Sphero module design..33
Figure 15 - PS3 manager module design...35
Figure 16 - Shows pseudo code of the tracking algorithm 44
Figure 17 - image → filter → mask → position process...................................45
Figure 18 - Filter containing noise...46
Figure 19 - Coordinate system used by the tracker...47
Figure 20 - Traceable object..47
Figure 21 - Data flow Sphero Object 50
Figure 22 – Sphero calibration 54
Figure 23 - Client - Sphero RRT 58
Figure 24 - Graph of streaming speed 59
Figure 25 – Graph of FPS Tracking 60
Figure 26 - Graph of CPU tracking...61
Figure 27 - Follow virtual dot test 63
Figure 28 - Plotting of round trip samples 64

Figure 29 - GPU vs CPU processing (figure from)..67

IV. List of tables

Table 1 - Client --> Sphero packet format 21
Table 2 - Client → Sphero packet description 21
Table 3 - SOP2 bit options 21
Table 4 - Sphero → client response packet 22
Table 5 - Sphero → Client response description 22
Table 6 - Sphero asynchronous packet structure 23
Table 7 - Sphero asynchronous packet types 23

Chapter 1 - Introduction

1.1 Overview

The word *Robot* arrives from the old Czechoslovakian word *robota* / *robotnik* meaning a slave or a servant[1]. A robot can be described as a programmable, self-controlled device capable of carrying out series of actions automatically, especially actions programmable by a computer.

Universities have used Robot technologies for outreach and recruitment purposes over the last decades [1], [2], [3]. The computer science (CS) department at the University of Tromso (UIT)[2] has in the later years invested in different robotic platforms for use in own their recruitment program. The recruitment program is responsible for targeting high school students and open their eyes for the world of computer science.

The use of robots in recruitment is an approach to give a demonstration of what simple programming skills can achieve. Small robotic devices is a good tool for this purpose because they have the effect of easily capture people's attention, especially when they seem to act intelligently in what they do [4]. A robot allows the audience to observe and interact, and stands out from more traditional demonstrations (e.g. talks, example systems, graphical visualizations).

The robot technologies used by UIT include Lego's robotic platform Mindstorms[3], Quadcopters[4] and the recently added robotic ball Sphero [5]. Sphero is a remotely controlled robotic ball developed by the American

[1] http://inventors.about.com/od/roboticsrobots/a/RobotDefinition.htm (accessed 11.05.14)
[2] http://uit.no/startsida (accessed 07.05.14)
[3] http://www.mindstorms.lego.com (accessed 14.05.14)
[4] http://quadcopterhq.com/what-is-a-quadcopter/ (accessed 01.06.14)

company Orbotix[5]. Inside the spherical shaped body of the device (see page 13 Figure 2) there is an internal core named the internal measurement unit (IMU). The IMU is similar to a miniature Segway[6], and it enables movement with two wheels that "drives" inside the enclosing hull of the device. Sphero contains different types of sensors (e.g. Gyroscope, accelerometer) and it uses a RGB LED to illuminate itself in various colors. iOS[7] and Android[8] devices are the mostly used platform to control the Sphero devices and there currently exists many different applications for the device[9].

One of the big limitations addressed by the administrators of the recruitment program is the robotic devices lacking possibilities in navigation and positioning. The devices used by the university are missing sensors that can obtain data about their accurate position and the position of devices in proximity. The administrators has posted request for a platform that would allow for position tracking of the devices. The system would make it possible to create richer and attention-grabbing applications for recruitment events and school visits.

This thesis addresses this request and presents Sphero NAV, a camera based navigation and control platform for the robotic ball Orbotix Sphero [5]. Sphero NAV is a Python [6] library that serves as a base for developing new Sphero applications. The library allows for communication and control of one or multiple Sphero devices using Sphero's rich API [7] and functionality. Sphero NAV implements a tracker system that uses image based position tracking. A web camera mounted in the celling over the tracking area captures a video stream. The system analyzes the images and locates the devices position.

Sphero NAV is a simple but efficient image-based position and control system and developers can use Sphero NAV to create different applications that utilize this functionality.

---

[5] http://www.gosphero.com/company/ (accessed 29.05.2014)
[6] http://www.segway.com/ (accessed: 20.04.14)
[7] https://www.apple.com/no/ios/ (accessed: 20.04.14)
[8] http://www.android.com/ (accessed: 20.04.14)
[9] https://play.google.com/store/search?q=Sphero (accessed 14.05.2014)

1.2 Problem definition

From the problem definition of this thesis:

> *"Develop a navigation platform for one or more users to control one or more robots (drones, sensor etc.). The platform should be easy to use and has to allow robots to operate on different levels of autonomy. The platform should also be easy to deploy and use both in the lab and when visiting schools and recruitment fairs."*

The interpretation of the problem definition lead to the project of creating a system that would allow its users to develop applications that controlled Sphero devices. The system would implement a positioning system that allows the application to obtain the position of the devices inside a given area. The choice of using Sphero was natural because UIT has just added it to its repository of robotic devices and Sphero had a well-documented API and functionality suited for a System like Sphero NAV.

1.3 Envisioned system

The envisioned system (Figure 1) of Sphero NAV had two components: The first was a positioning module that tracked devices by using images obtained by a camera. The system would analyze and find the (X, Y) coordinates of each device. The second component was a Sphero library that would implement the core functionality from Orbotix Sphero API [7] including communication with devices, necessary calibration and functionality for use with the tracker module.

The envisioned system would make it possible to control and receive information from the Sphero devices. This included access to sensor data and operational commands (e.g. Movement, light controls).

Implementing Sphero NAV as a software library allows for other applications to utilize the retrieved positioning and sensor data from the Sphero devices.

The two main tasks of the system were:

a) Implement a fully functional python library for controlling and communication towards the Sphero devices.

b) Implement an image based tracking system that tracks the (X, Y) coordinates of the device. Coordinates will be obtained by analyzing images from a live video stream.

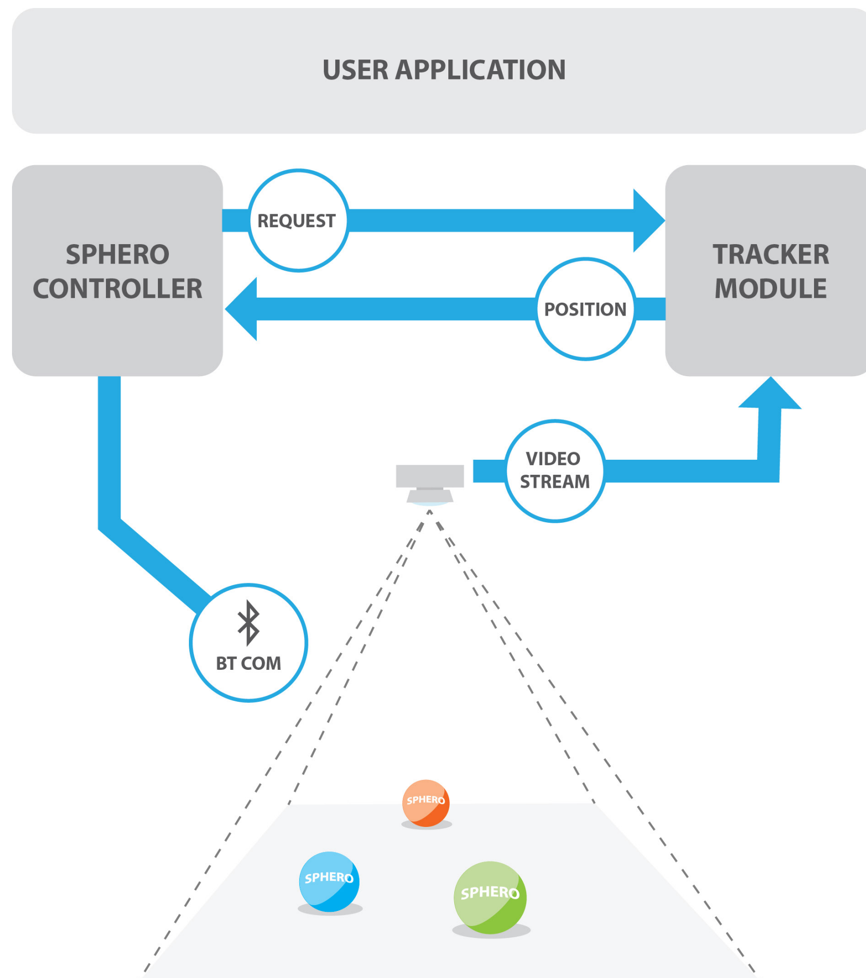


Figure 1 - Conceptual architecture

1.4 Contributions

The contributions made from this thesis are:

- A Sphero control API for Python
- A tracking system that allows for tracking of the Sphero devices
- A PS3 controller module that allows application to take use of game controllers.
- Evaluation and discussion of the Sphero NAV system
- A Video demonstrating usage of the Sphero NAV system [8]

1.5 Limitations

In the work of this project, some areas are not taking into account.

There has been no focus on the security aspect of the system. Potential security threats in the implementation of the system has not been evaluated and the current implementation as not been optimized to handle any potential threats that may exist.

1.6 Outline

The organization of the remainder of this thesis is as follows:

Chapter 2: Presents related work to the Sphero NAV system
Chapter 3: Present background material for the Sphero device
Chapter 4: Presents Sphero NAV's architecture, design and usage
Chapter 5: Presents implementation details of the Sphero NAV system
Chapter 6: Evaluates and discusses the Sphero NAV system
Chapter 7: Concluded this thesis and presents future work

Chapter 2 - Related work

2.1 Introduction

This chapter presents work that is related to and/or has similarities with the Sphero NAV system.

2.2 Bouncing Star

Bouncing star [9] is an entertainment system developed at the University of Electro-communications in Tokyo Japan (see videos [(10,11)]). Bouncing star present a gaming platform that uses a self-developed spherical input device called a smart ball. A smart ball is a device in the same shape and size of a tennis ball and contains sensors, communication, computational power and lights. The developers behind bouncing star have created different ball-based games suited for the smart ball under the name "digital sports". To add a new dimension to the games, the system uses sensor readings, graphical visualizations (CGI[12]) and the onboard LED's to enhance the user experience.

In comparison with Sphero NAV, Bouncing star is an input system where smart balls are moved physically by the end-users. Sphero NAV is different and is intended for the end-user to control Sphero devices with game controllers or watch them perform different visualizations. Note that there exists Sphero applications that use the Sphero device for input, and by using Sphero NAV's data streaming support this kind of support could be implement in new applications. Bouncing Star is used on academic conferences and museums but it has also been used for scientific experiments.

[10] http://www.youtube.com/watch?v=rZxLO77dtho (accessed 16.05.14)
[11] http://www.vogue.is.uec.ac.jp/project/projects-1/bouncing-star (Accessed 14.05.14)
[12] http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Computer-generated_imagery.html (accessed 31.05.2014)

The smart ball used in Bouncing Star has similarities to the Sphero device, but holds no internal actuators for movement. Whereas Sphero uses Polycarbonate for its body, the bouncing star device is made of a rubber like finish making the bouncing star bouncier and more suited for ball games. The developers behind Bouncing star describes the smart ball as an organic user interface [10]. An organic user interface is an interface that uses a non-planar shaped object as a display for its primary output and input.

Bouncing star tracks the position of the smart ball by using a fixed camera, this is similar to the tracking system implemented in Sphero NAV, and the main difference is that Bouncing Star uses an Infrared camera. The tracking works by tracking IR light emitted from the onboard IR LEDs in the smart ball. IR light is not visible to the human eyes and makes it possible for the devices to display different colors from the normal RGB LED's without affecting the result of the tracking. This is not possible with the Sphero because it does not hold IR diodes. The disadvantage with IR tracking is that direct sunlight would affect the result and make it impossible to track the devices.

The Bouncing star system serves graphical support. A projector is placed in the celling pointing down on the area assigned for the smart ball. The projector displays different types of game graphics affected by the usage of the smart ball. Combined with sound effects this creates a richer experience for the end users. The current implementation of Sphero NAV does not implement this support, but it is noted for one of the things to add in the future.

2.3 Ping pong plus

Ping pong plus (PPP) [11] is a research project from MIT Media Laboratory. PPP is a gaming system built on top of a standard ping pong table (see video[13]). The motivation behind the ping pongs plus project was the interest in designing a system that sets the use of physical movement from the end users in focus. The goal is to remove the requirement of using standard input devices such as mouse, keyboard or joystick.

Ping Pong Plus is a digitally enhanced version of the ping pong game, and the system is implemented in two parts. A tracking module used for tracking of the

[13] http://www.youtube.com/watch?v=AZO8sfmpKIQ (accessed 19.05.14)

ping pong ball, and a graphic system where a projector is placed over the pool table and used for displaying in game graphics.

The tracking of ping pong plus is achieved by capturing sound using 8 microphones. The microphones are used to track the sound made from the touchdown of the ping pong ball. A tracking algorithm calculates the position of the touchdown by calculate the different time differences in the captured sound. This is similar to how humans detects location of the sound.

Ping Pong plus has no direct similarities with Sphero NAV, but it is interesting to see how they perform the tracking of the ping pong ball. The Ping pong equipment used in PPP is built upon a standard pool table, and the ball holds no extra technology for enabling tracking. Tracking is performed by differences in the captured sound. This approach is probably not suited and possible for tracking of Sphero's.

One of the things used by Ping Pong Plus that Sphero NAV could use is the graphical system where a projector is used to display graphics. This is something used by all the related systems mentioned in this chapter.

2.4 Open Pool

Open pool (see video[14]) is an open source project that implements an interactive entertainment system for pool[15] tables. The system uses two Microsoft Kinects and pocket detectors (the holes in the pool table) to track the position and state of the billiard balls. A projector mounted in the celling is used to project graphics on the mat of the table. The system is used to give an interactive experience when playing pool. A depth camera[16] onboard the Kinects is used to obtain the position of the billiard balls. Open Pool focuses on the graphical experience. The system does not use any special billiard balls or ques.

Open Pool uses Microsoft Kinect for tracking the position of pool balls. A depth camera is used for finding the tracked distance to every object. This data can be used to extract objects on a flat ground. In the early development phase of

[14] http://www.youtube.com/watch?v=e3Ywdw8IuG8 (accessed 19.05.14)
[15] http://www.theworldgames.org/the-sports/sports/precision-sports/billiard-sports (accessed 30.05.14)
[16] http://www.youtube.com/watch?v=uq9SEJxZiUg (accessed 16.05.14)

Sphero NAV the idea was to use the same approach and combine the depth data with the captured image to achieve a more accurate tracking result. This approach worked relatively well on short distances, but when the Kinect was placed high above the tracking area, the depth data was too inaccurate to locate the Sphero's on the ground. The Kinect was also challenging to setup and use from python. The tracking of Sphero NAV is for these reasons based on image tracking alone.

2.5 Pixelbots (Display swarm)

Pixel boots [4] [12] [13] [14] is a research project in cooperation with Disney[17]. Pixel boots is a robotic platform that implements a display swarm. A display swarm is a swarm of small robotic devices where each device presents a pixel used to create a larger image. Each device in a swarm has the opportunity of displaying different colors. Patterns are created by having the devices placed in various positions. The current version of a pixel bot is a small circular robot with a LED light in the top. The devices are custom built and designed to move in a planar ground. The devices use magnetic wheels making them possible to use on avertical magnetic plane.

The pixel boot devices are tracked in a similar approach has the smart ball presented by Bouncing star, and each pixel boot holds IR lights that are tracked by a overlooking fixed camera. Pixelbots has also implemented functionality for using a projector to enhance the visualization of the system.

Typical uses of display swarm are on concerts, sports events, amusement parks etc. The pixel bot project has also researched on making a display swarm with small remote controlled helicopter like devices. This would allow for using a 3D space to create patterns. But this part of the project is still in the early phase.

The Sphero devices are similar to the robots used by pixelbot. Each Sphero has the possibility of displaying custom colors with their onboard RGB LED light and it would probably be possible to create a display swarm application with the tracking system provided by Sphero NAV. One of the limitations with the Sphero is that it uses Bluetooth for its communication. A Bluetooth network[18]

---

[17] http://www.disney.no/ (accessed 31.05.2014)
[18] http://www.informit.com/articles/article.aspx?p=21324 (accessed 29.05.2014)

(Piconet) has a limitation of only communicating with up to seven devices at once. This limits the amount of Sphero devices possible to communicate with from one adapter. Pixelbots uses RF communication and writes that they successfully support 100 devices with an update rate of 10Hz. The communication limitation is based on Sphero devices hardware and is not possible to improve without physical changes performed by Orbotix.

Chapter 3 - Orbotix Sphero

3.1 Introduction

This chapter goes into the details of the robotic device Sphero [5] [15], the Sphero API and communication protocol. It describes its possibilities and supported functionality.

Figure 2 - Shows the Sphero device

3.2 The device

The Sphero is a ~150 USD spherical robotic ball created by the American company Orbotix[19] and its co-founder Ian Bernstein describes the Sphero as the next generation of gaming systems.

The device consists of an internal core with similarities to a tiny Segway[6]. The core (Figure 3) holds two electronic motors, a RGB led light, accelerometer and gyroscope and is controlled over a Bluetooth connection [16]. The core runs inside a shock resistant and watertight spherical Polycarbonate[20] housing. The movement of the device is accomplished with the same principle as a hamster running inside a hamster wheel. The two motors are used to control the movement and heading of the device. A counterweight (induction coil used for charging) placed in the bottom of the core and a stabilization algorithm processing data from the onboard sensors holds the core in a horizontal position. The Sphero is charged using an induction charger.

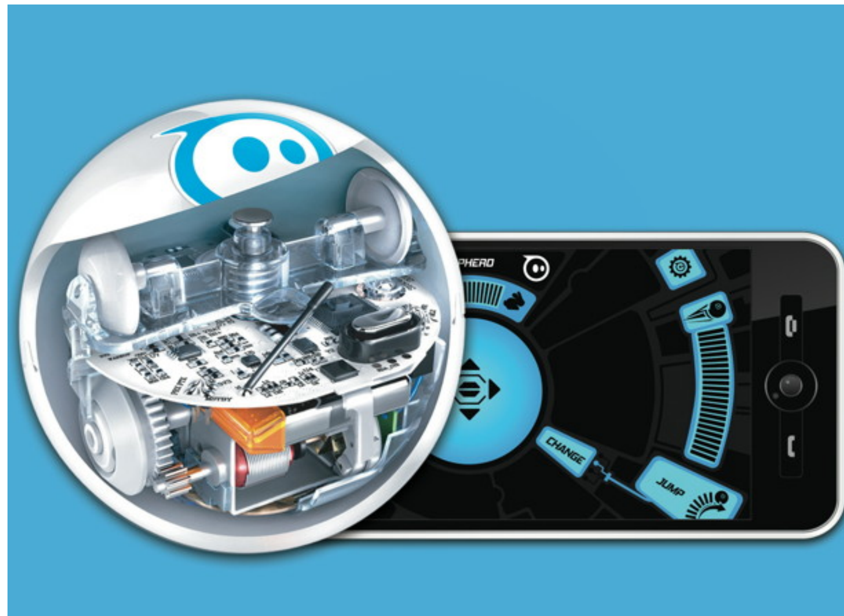


Figure 3 - Shows the inside and IMU of the Sphero

[19] http://www.gosphero.com/company/ (Accessed 21.04.2014)
[20] http://www.bpf.co.uk/Plastipedia/Polymers/Polycarbonate.aspx (Accessed 21.04.2014)

The housing of Sphero is very robust and made to withstand hard collision and drops. The device is approved to safely handle droops from up to 0.5m. The actual limit is probably much higher, and one article[21] claims that the Sphero successfully handled a fall of 7.6 meters without damage. The Sphero housing is waterproof and the device has good buoyancy, making the Sphero able to float and drive in water. The Sphero devices could be used with a rubber cover for gaining better traction. The covers comes in different colors and where used in Sphero NAV for one approach of tracking the devices.

There exist a wide range of applications for controlling and doing different things with the Sphero devices. Orbotix creates many applications for Sphero themselves, but since Sphero has such a well-documented API, it seems that it is very popular for third party developers to create applications as well. This leads to a wide choice of applications. Orbotix is also active in hosting hackatons where developers are encouraged to create new Sphero applications.

3.2.1 SDK's

Orbotix offers SDK's [17] for different platforms. The currently provided SDK's are android, iOS and windows phone. There are unofficial SDK's provided by third party developers including support for Windows8, Node and Ruby. There also exist an SDK for python [18]. The python API is in its early alpha stages of development and the API was used as a base for the Sphero library implemented in Sphero NAV.

[21] http://electronics.howstuffworks.com/sphero1.htm

3.3 System Design

3.3.1 Client – Server

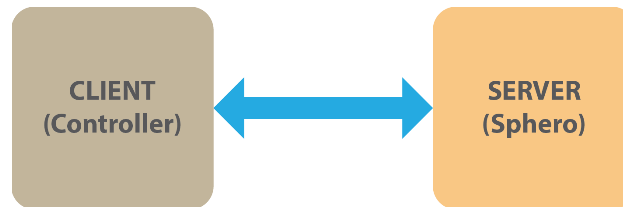


Figure 4 - Sphero client --> server model

The communication between the user application and the Sphero device is performed in a client server fashion [19] (Figure 4). The Sphero device acts as server where the clients connects and send synchronous messages and receive responses. Sphero sends asynchronous data back to the clients when one or more of the asynchronous streaming features it supports are activated by the client (see Table 7).

3.3.2 Virtual devices

Sphero divides its internal responsibilities into several virtual devices. Orbotix says that this division was implemented to make the separation of task more clear [7]. Typical virtual devices of the Sphero are: the control system, the bootloader and the orbbasic device. The control system handles all the commands that control the hardware on the device (e.g. heading, speed, lights). The bootloader is responsible to handle firmware downloads and other core functions. The OrbBasic (see 3.4.4) interpreter is used to download and run user created OrbBasic programs on the device. A bitfield set in each package to the device (Table 2) specifies witch virtual device the packet is intended for.

3.4 Sphero Overview

Sphero is equipped with a gyroscope and accelerometer, the data from these sensors including motor information is accessible by the users. Data can be accessed in raw and aggregated formats.

A gyroscope is a sensor that keeps track of the orientation of the device. An accelerometer measures the acceleration forces. These sensors are used to give valuable information about the movement of the device. Figure 5 shows what the gyroscope and accelerometer can measure.

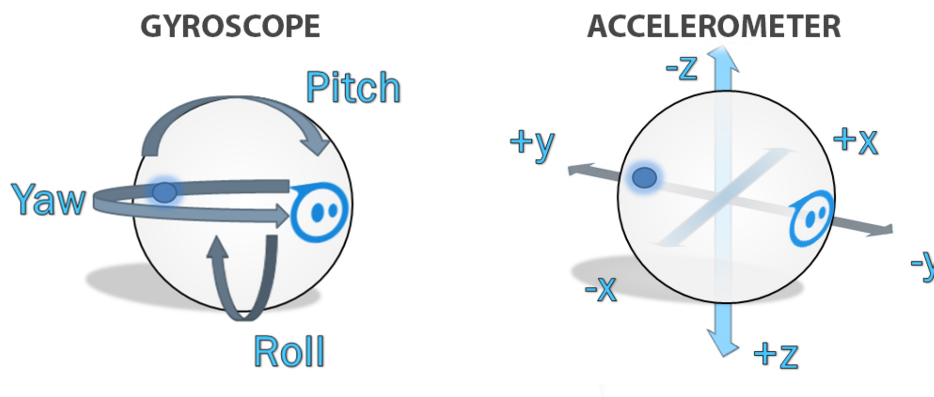


Figure 5 - Accelerometer and Gyroscope in Sphero

The Sphero is at is base a simple device, and at its most basic level the hardware implements a small set of raw inputs and outputs [7].

- **Raw Inputs**
  - Three axis rotation rate gyro
  - Three axis accelerometer
  - Approximate ground speed through motor data
  - Data from radio link
  - Battery voltage

- **Raw Outputs**
  - Power to left and right drive wheels
  - RGB LED color value
  - Back LED intensity
  - Data to radio link

The internal software inside the device aggregates and uses these raw hardware I/O elements to construct higher level data systems that are useful for the application controlling the device. Some of these systems are: heading control, distance measurement, collision detection, virtual locator system, data integrators[22]/differentiators and more.

3.4.1 RGB light

A RGB LED is a light source that consists of three light emitting diodes (LED's) in the color: red, green and blue (RGB). By adjusting, the brightness of each individual LED's it is possible to create a wide gamut of colors. This approach for creating different colors is the same as used when mixing colors for painting.

The Sphero device is equipped with a RGB LED that illuminates the upper part of Sphero's body. This allows the device to "glow" in different colors. The color and intensity of the RGB LED is controlled with commands from the Sphero API.

The tracking system of Sphero NAV allows for tracking of objects with different colors. The Sphero's can dynamically change its body color and this property can be used when searching for Sphero's in an image.

3.4.2 Coordinate system

Sphero uses a coordinate system for movement where angle 0° is equal to drive straight forward in what would equal along the Y-axis positive direction in a Euclidean coordinate system. The angles are positive in the clockwise direction. This is different from the Euclidean coordinate system where angle 0° is straight down the positive X-axis and an angle of 90° would be along the Y-axis. Figure 6 shows a comparison between the Euclidian coordinate system and the one used by Sphero.

Sphero NAV uses the Euclidian coordinate system for all of its calculations. Every command that includes use of the coordinate system from Sphero is therefore translated to and from the different coordinate systems when communication with the Sphero device. The use of Euclidean coordinates in

[22] http://www-01.ibm.com/software/data/integration/ (accessed 27.04.14)

Sphero NAV was used to make it easier to perform calculations using existing mathematical libraries.

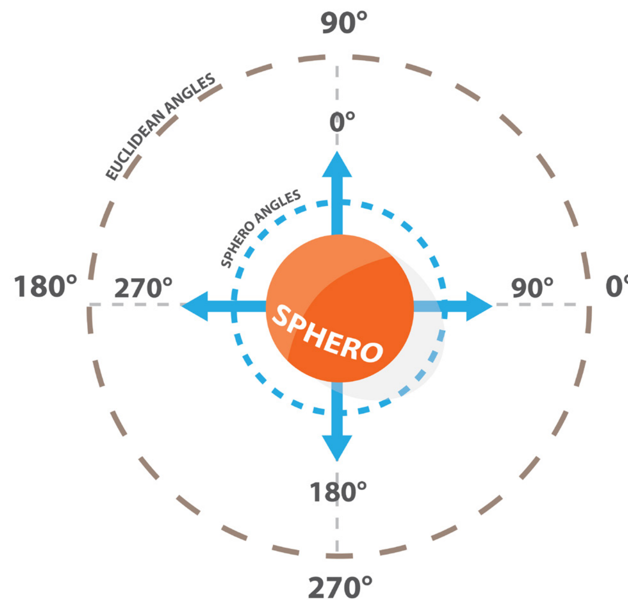


Figure 6 - Sphero angles vs. Euclidean

3.4.3 Locator

Sphero holds a service called the Sphero locator [20]. The Sphero Locator is a system that implements an onboard positioning system to keep track of Sphero's movement. The locator uses a virtual planar two-dimensional space to represent Sphero's current position. The position is relative to the position from the startup of the device and the x, y coordinates are measured in cm.

The locator service serves data about Sphero's current position and velocity inside the two-dimensional virtual space. The location is calculated by aggregating data from Sphero's onboard sensors. The locator service is relatively accurate, but it is sensitive to collisions and the results get inaccurate after some time. Sphero's API allows for operations for booth setting and getting location data.

The first idea when developing Sphero NAV was to use the locator data and combine it with the tracking data from Sphero NAV's tracking system. There was not enough time to implement this feature. Note that Sphero NAV holds support for retrieving and setting data from the locator, and even though it's not implemented in the tracker system, it still allows the application developers to use the locator data if needed.

3.4.4 OrbBasic

Orbotix has developed an interpreter for running code snippets directly on the device. This functionality is called OrbBasic [21] and it implements a simple Basic[23] like interpreter. This thesis does not use this functionality of Sphero, but it could be extended in the future to also support it. Orbasic allows users to create programs that are executed on the Sphero Devices.

3.4.5 Macros

Sphero implements functionality for running Macros [22]. A macro is a set of Sphero commands transferred and executed on the device in bulk. This functionality allows for the end-users to create a macro the Sphero can execute when asked to. This feature allows for complex operation on the Sphero with minimal data transfer between the Sphero and client. Although this functionality could be used in Sphero NAV, there was not enough time to implement support for it. Note that the design of the Sphero NAV library makes it possible to add support for macros in the future.

3.5 Packet structure

Sphero comes with a well-documented communication protocol. This section goes into the details of the packet structures used for communication with the Sphero.

3.5.1 Synchronous packets

The client sends synchronous command packets to the Sphero device. Synchronous packets are used for all communication in the clients → sphero direction. The normal packet flow is that the client sends a request to the Sphero device and receives a response packet for this request. Each request holds a sequence number in the range of 0-255 that is sent back in the

[23] http://www.computerhope.com/jargon/b/basic.htm (accessed 18.05.14)

response packets. Responses from the Sphero can be deactivated by setting a flag in the request packet.

Request packets sent to the sphero includes the necessary data for performing the command on the device. The format of this data is specified for each command in the Sphero API.

The request packets are sent in the following format:

Client → Sphero packet format:

SOP1	SOP2	DID	CID	SEQ	DLEN	<data>	CHK

Table 1 - Client --> Sphero packet format

Meaning of each field:

SOP1	Start of packet #1	Always 0xFF
SOP2	Start of packet #2	Per-message option (see Table 3 - SOP2 bit options)
DID	Virtual Device ID	The virtual device this packet is intended for
CID	Command ID	The id of the command
SEQ	Sequence number	The sequence number of the packet. 0x00 to 0xFF. Used in the response packet
DLEN	Data length	The length of the data in this package
<data>	Data	The data for the command
CHK	Checksum	The modulo 256 sum of all the bytes from the DID through the end of the data payload, bit inverted (1's complement)

Table 2 - Client → Sphero packet description

SOP2 flags:

Bit 0	Answer	When set to 1, send reply to this packet
Bit 1	Reset timeout	Reset the Sphero inactivity timer.
Bit 2-7	Future use	Always set to 1

Table 3 - SOP2 bit options

3.5.2 Synchronous responses

Every request sent to the Sphero results in a response packet back to the client (unless disabled). The response packet holds a response code that holds the status of the request. (Successful or not).

Sphero groups its commands into two categories. Set and Get commands. Set commands assign some internal state on the device whereas Get commands get some state or data from the device. The Set commands receives a response type that is defined as a simple response. The simple response is in the format as is displayed in Table 5 and holds no data. The Get responses are used to send data from the device to the client. The format of the data for each response is specified in the Sphero API. (e.g. sensor data, battery state) and varies for each response.

The response packets are sent in the following format:

SOP1	SOP2	MRSP	SEQ	DLEN	<data>	CHK

Table 4 - Sphero → client response packet

SOP1	Start of packet #1	Always 0xFF
SOP2	Start of packet #2	Set to 0xFF when this is a
MRSP	Message response	Response Code Successes, failed e.g.
SEQ	Sequence number	The sequence number of the request packet this response belongs to
DLEN	Data length	Length of the response data
<data>	Data	Data of the response
CHK	Checksum	The modulo 256 sum of all the bytes from the DID through the end of the data payload, bit inverted (1's complement)

Table 5 - Sphero → Client response description

3.5.3 Asynchronous packets

The Sphero API implements support for asynchronous package streaming. The streaming of these packages is activated/deactivated with different synchronous commands from the client. Typical asynchronous packets would be a notification that the Sphero has collided [23], sensor data, battery levels etc. The asynchronous packet types supported in Sphero's current API are listed in Table 7.

SOP1	SOP2	ID CODE	DLEN-MSB	DLEN-LSB	<data>	CHK
0xFF	0xFE	Packet type	<msb>	<lsb>	<data>	checksum

Table 6 - Sphero asynchronous packet structure

ID CODE	DESCRIPTION
0x01	Power notification The current voltage on the device
0x02	Level one Diagnotics. Send a string of device information.
0x03	Sensor data streaming
0x04	Configuration block content.
0x05	Pre-sleep warning
0x06	Macro markers
0x07	Collision detection
0x08	OrbBasic print msg
0x09	OrbBasic errer message, ASCII
0x0A	OrbBasic error message, binary
0x0B	Self level result
0x0C	Gyro axis limit exceeded
0x0D	Sphero soul data
0x0E	Level up notification
0x0F	Shield damage notification
0x10	XP update notification
0x11	Boost update notification

Table 7 - Sphero asynchronous packet types

Chapter 4 - Sphero NAV

4.1 Introduction

This chapter presents Sphero NAVs architecture and design. It ends by outlining use cases and some code examples of how to use the system.

Figure 7 - Sphero NAV logo

4.2 Architecture

A software library is a library that adds extended functionality for the application developer. Sphero NAV is a software library that allows for position tracking and controlling of the Orbotix Sphero. A well written software library should be easy to use, works flawlessly and provide detailed error information [24].

Sphero NAV's architecture uses a modular design[24] allowing developers to use some or all of its provided functionality. The separation of concerns divides the architecture (Figure 8) vertically into three different software modules: a *Tracker module* serving object tracking, a *Sphero module* for using and controlling Sphero devices and a *PS3 module* that implements support for game controllers. The software modules can be used together or separately. Sphero NAV also provides a simple utility library that holds useful tools for the application layer to utilize.

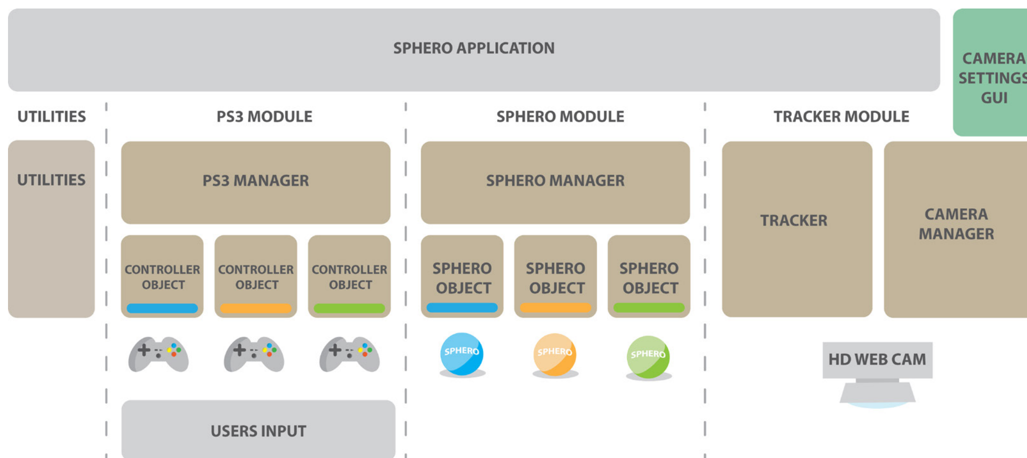


Figure 8 - Architecture of Sphero NAV

[24] http://msdn.microsoft.com/en-us/library/gg405479(v=pandp.40).aspx (accessed 29.05.2014)

The Sphero module consists of two tiers: a manager layer and a Sphero object layer. The Sphero module allows applications to take use of and control Sphero devices. Each Sphero device is connected and controlled through a *Sphero object*. A Sphero object implements the interface for communicating, data streaming and control of the Sphero.

The Sphero manager is used to manage multiple Sphero's and provides easy search and discovery services of nearby devices. Sphero objects are passed to the application layer via the Sphero manager.

The tracker module is used in applications where position data of devices is needed. The tracker handles video capture and object tracking. The camera manager is used for configuration of the connected camera.

The architecture of the tracker module is divided vertically into two components, where the tracker holds all functionality for tracking and the camera manger is used for configuring settings on the camera. The two components hold no connections to each other.

The PS3 module uses the same two-tiered architecture as the Sphero module. The PS3 module allows applications to take use of input from one or multiple PS3 game controllers. Each game controller is connected through a *Controller object*. A controller object allows the developer to map application functionality to the interface of each controller with registering callbacks. The PS3 manager holds an event-handler and events from the controllers are passed to the application in the form of callbacks.

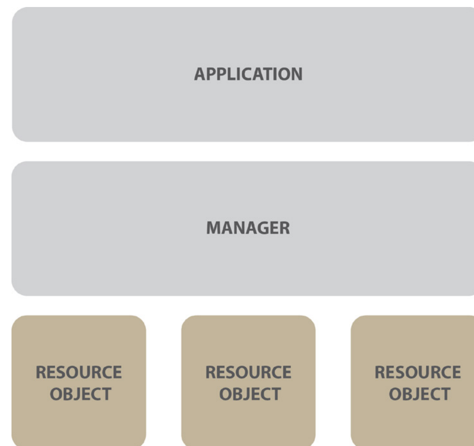


Figure 9 - Manager Design pattern used for Sphero and PS3 module

The architecture of both the Sphero and PS3 modules consist of a manager layer. The managers makes it easier for developers to use each resource, and the need of boiler plate code for getting the system up and running is reduced (e.g. searching, event handling). The manager design (Figure 9) allows for handling of the different resource objects (Sphero's and PS3 controllers).

The vertical division of Sphero NAV's architecture into separate modules enforces a separation of functionality for each respected field (e.g. Sphero, Tracking, PS3 controller). All code for each subject is held inside its respected module and the modules holds no direct coupling to each other. This architecture was used to make it easier for developers of Sphero NAV to extend, modify and add new modules in the future.

4.3 Design

4.3.1 Tracker

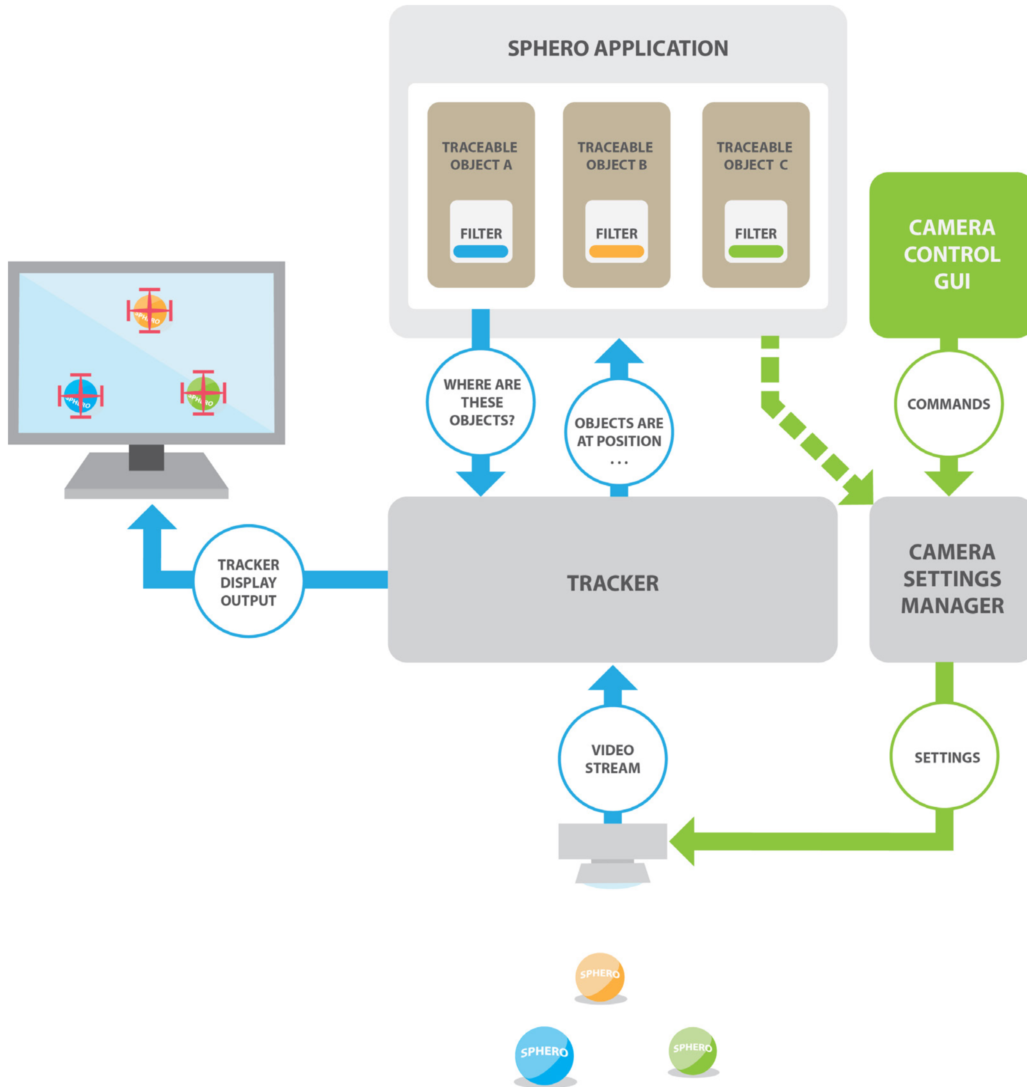


Figure 10 – Tracker design

The tracker implements support for applications to retrieve position of devices inside a tracked area. The tracker supports tracking of multiple heterogeneous devices. A requirement is that a Traceable Object represents each device tracked. The application would activate a tracking of objects by using the trackers method *track_objects* (5.3.1). *track_objects* takes a list of traceable object as parameter. The position of each traceable object is found in the

tracker by an algorithm based on image analyzing (5.3.1). After the objects position is found, the result is returned back to the application. Note that traceable objects are passed to the tracker on every tracking call. This design allows the developer to determine the objects to track for each tracking call.

Traceable objects are designed to serve as containers for storing tracking data and includes a tracking API. The tracking API is the interface for accessing tracked position, time, speed and direction and allows the application to access data from previous tracking's.

A filter held in the traceable object implements logic for distinguishing a device in an image (e.g. color, shape). The filter is used to create a tracking masks (see Figure 11 and Figure 17) used by the tracker. Masks are used in the process for determining the position of each device (5.3.1). The tracker supports different types of filters and the system is designed so developers can create custom filters for distinguishing different objects by inheriting the filter class.

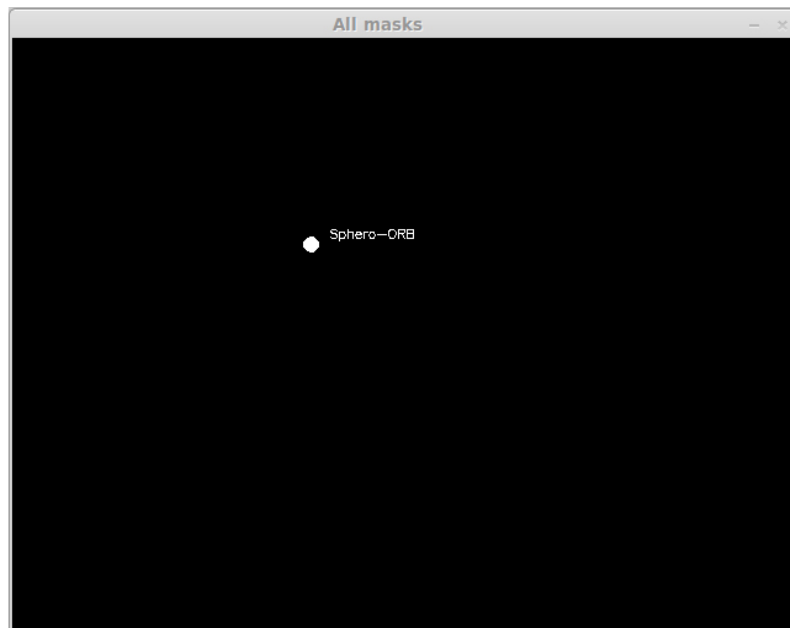


Figure 11 – Tracking mask displayed by the tracker

The design with Traceable objects and filters was used to make the system configurable and dynamic. This design allows developers to override and extend functionality to suit their particular application.

Traceable Sphero is an object that extends the interface of the traceable object. Traceable Sphero serves as a "bridge" between the Sphero and tracker module and holds functionality used exclusively for tracking Sphero's. Additional functionality in Traceable Sphero includes Sphero Calibration (5.4.4) and support for drawing Sphero related graphics to the tracker display (e.g. sensor data, device name) (see Figure 13).

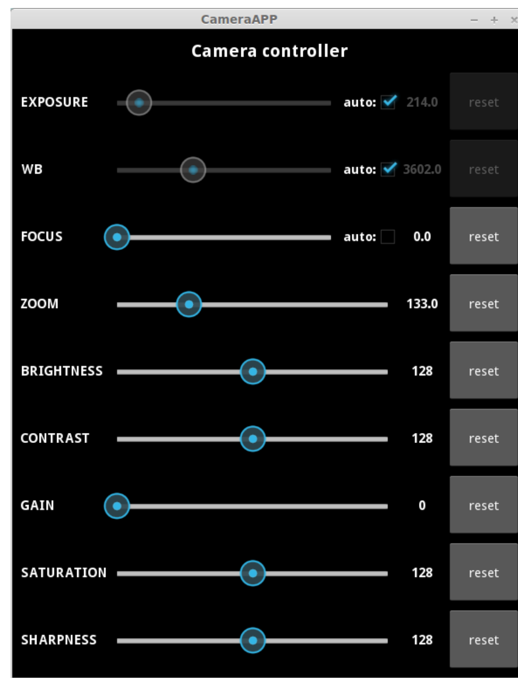


Figure 12 - GUI of the camera settings manager

A camera positioned over the tracking area captures a video stream used for tracking of objects. The camera manager allows developers to control the settings on the camera (e.g. white-balance, focus, zoom, exposure). Sphero NAV was designed for use in different locations and factors like lighting and the placement of the camera differs and affects the capture. The camera manager has two possibilities for usage: first, a simple GUI (Figure 12) for manual adjustment. Second, an API that allows for control of camera settings directly from the application (Figure 10). The current version of the settings manager is limited to the Logitech C920 web camera[25]

[25] http://www.logitech.com/no-no/product/hd-pro-webcam-c920 (accessed 29.05.2014)

The tracker serves a graphical user interface (GUI) to displays the video stream and tracker related graphics. The tracker module includes a graphic library allowing developers to draw graphic on the tracker GUI. This functionality is accessible through the traceable object class. The default graphics implemented in Sphero NAV includes graphic for the position and direction of each object, FPS and device information (e.g. sensor data). The masks used for the tracking is displayed in a separate window (Figure 11). This is useful for the users to determine if the masks are masking out the objects correctly. Figure 13 shows a screen dump from the tracker windows where two Sphero devices are tracked. The blue line shows the direction of the internal core of the Sphero (streamed from the Sphero) and the red line shows the tracked position and heading of the device. The white Sphero is stationary so it has no red line for indication its current direction of movement.

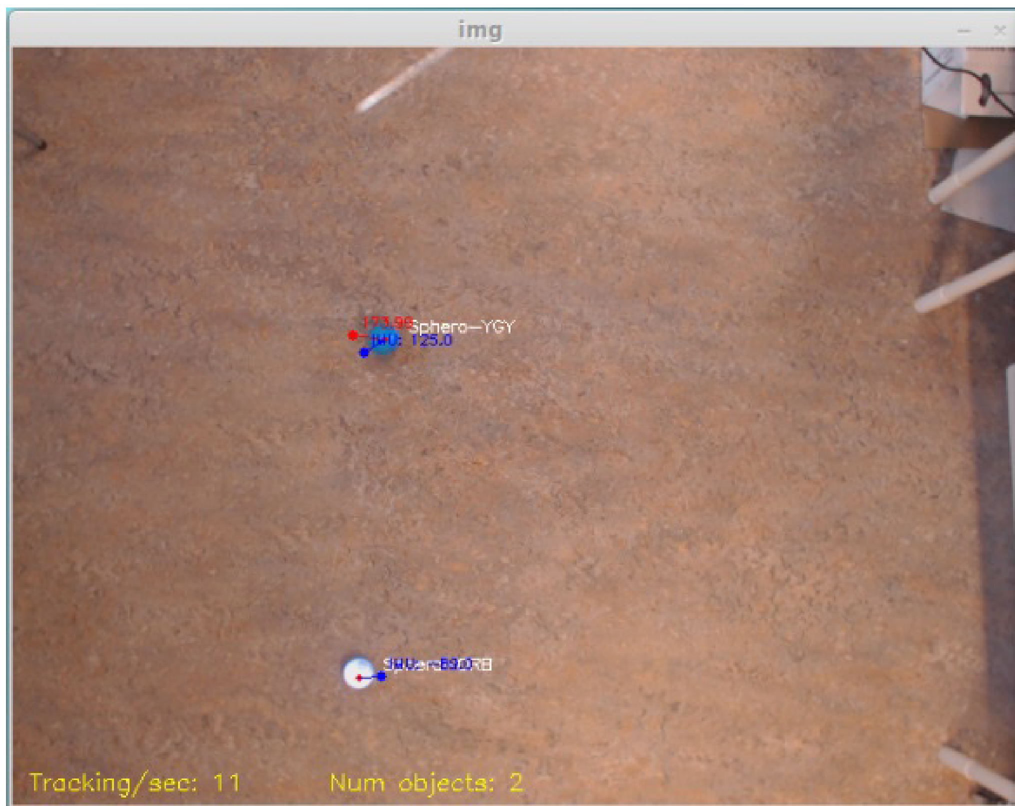


Figure 13 - Tracker GUI

4.3.2 Sphero Module

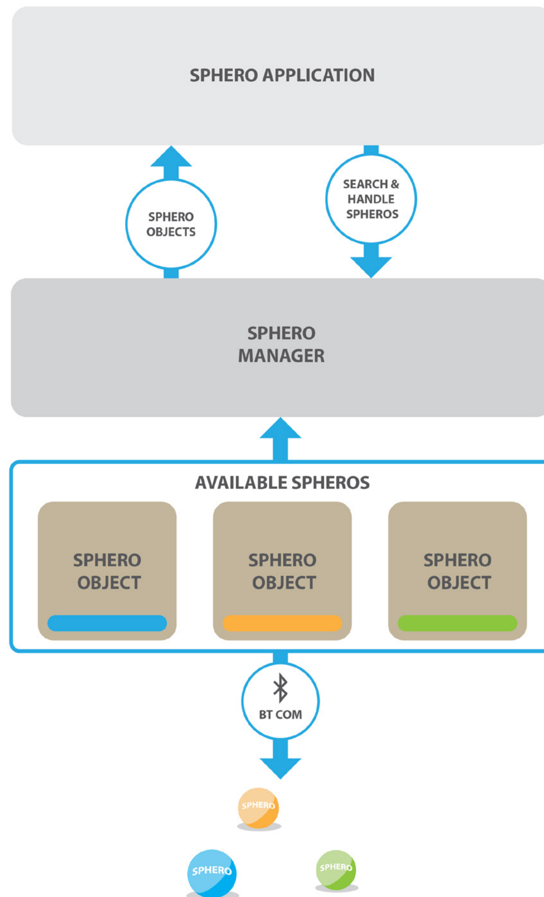


Figure 14 – Sphero module design

The Sphero Module holds functionality for applications to use and control Sphero devices. Every Sphero used in an application communicates through its own *Sphero object*. A Sphero object is the interface for all actions regarding a Sphero device. The Sphero objects implements methods for connecting, controlling, and receiving data. The design of the Sphero object was built upon the Python Sphero API library [18], but much of the code has been re-implemented. This was necessary for implementing support for, dynamic Sphero discovery, streaming, multiple devices etc. A Sphero object allows the application developer to access Sphero's functionality and it implements functions from the Orbotix Sphero API (e.g. Movements, lights, sensor

streaming). Due to the timeframe of this thesis, it was not possible to implement the full API from Orbotix. The core and most useful functionality from the Sphero API was therefore prioritized.

The sphero objects implements support for a streaming service where applications can receive data asynchronous from the Sphero (e.g. Accelerometer, gyroscope, collisions detection, battery level). Data streaming is used in application where data from the Sphero is necessary. The streaming interface implements some of the asynchronous functionality from the Orbotix Sphero API (Table 7). Data streaming is used by the application with registered callbacks that are triggered by the Sphero object whenever data is received from the device.

The design of using a Sphero Object for each Sphero is based on the code from the Python API [18]. The design of using Sphero Objects enforces that all logic for using a device is held in one place. This design makes the Sphero devices trivial for the developers to use.

The Sphero manager allows for the developer to easily search and use Sphero devices. The manager supports synchronous and asynchronous Sphero discovery. Synchronous discovery allows the developer to manually search for new devices. Asynchronous discovery allows the developer to register a callback that is triggered whenever a new device is found. Asynchronous discovery of Sphero devices runs in its own separate thread and is activated and started by the application (Example 2). In both cases of async/sync discovery a Sphero objects are served back to the application.

A vector controller is provided to control the Sphero devices in a more game like fashion. The vector controllers allow the developer to control the direction and speed with the use of a 2D vector class provided from the utility library. A calibration step (5.4.4) is used to align Sphero internal controls system with the coordinate system used by the tracker and the vector controller. This calibration is necessary when using the vector controller.

A successful calibration would mean that setting vector controller to *vectorController.speed.y = 255 would* results in the Sphero driving up the y-axis of the tracked image, and just not in some random direction.

4.3.3 PS3 Module

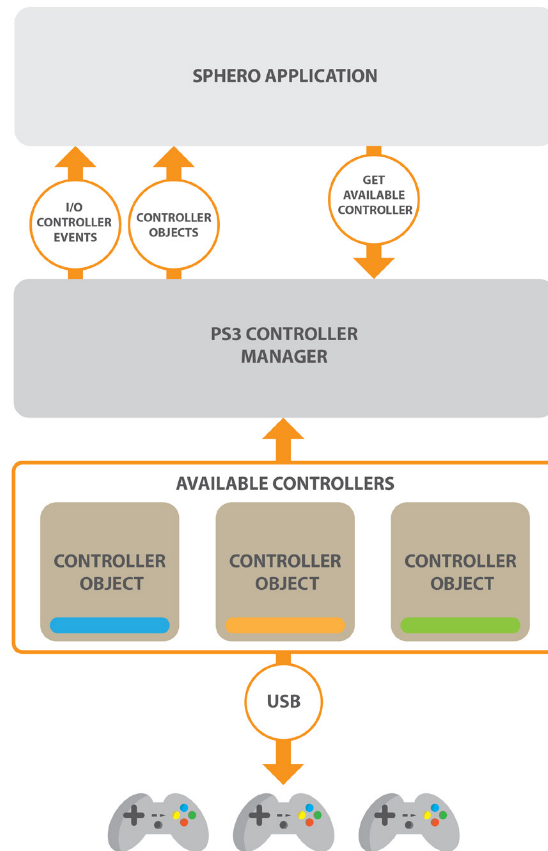


Figure 15 - PS3 manager module design

The PS3 module allows the application to take use of PS3 game controllers. Each game controller is connected with its respected *Controller Object*. A controller object has an interface that allows developers to register callbacks triggered on user input. A typical use case for a PS3 controller would be manual controlling of Sphero devices by the end-users.

The PS3 manager is used to serve controller objects to the application and holds an internal event handler. The event handler runs in a separate thread and receives input events from the connected PS3 controllers. When a button on a controller is pressed the callbacks registered in the responding controller object is called and the application is notified of the input. There are three

types of events supported for each button on a PS3 controller: On-press, on-release and on-axis. On-axis is used for returning the position of a joystick or the pressure used on a button. All events to a controller must be set on the same Controller objects, but it is possible to pass this object around. This allows the application to use the same controller in multiple places. The current implementation supports one callback per button event and no parameters for the callbacks. This is something that could be added in the future.

4.4 Use Case

4.4.1 Application ideas

Typical applications that would take use of the Sphero NAV library would be different types of games and visualizations. One of the early ideas for a fully autonomous Sphero NAV application was to implement Craig Reynolds famous Boids simulation[26, 27]. Each Sphero would represent a boid, and the application would control the Sphero devices to drive inside the tracking area in a herd like behavior.

For a semi-autonomous application, one idea was to create an application that would implement virtual borders. During recruitments fairs, virtual borders could be used to create a restriction, so manual control of the Sphero's was only possible inside the traceable area. A virtual border would stop runaway devices and restricting the demo to the designated area.

4.4.2 API usage examples

This section shows examples of how to use the Sphero NAV library. Since Sphero NAV is a software library, the easiest way to show its usage is by showing code snippets. The following examples shows working Python code and demonstrate how to use some of Sphero NAV's core functionality.

Note: *To simplify the examples no exceptions are handled.*

[26] http://www.red3d.com/cwr/boids/ (accessed 14.05.2014)
[27] https://www.youtube.com/watch?v=39Fktr5zaIY (accessed 30.05.2014)

• Synchronous device discovery and usage

```
1   import sphero
2
3   # Get Sphero manager
4   manager = sphero.SpheroManager()
5
6   # Get a device
7   device = manager.get_available_device()
8
9   # connect to Sphero
10  device.connect()
11
12  # Move Sphero full speed heading 0 for 2sec
13  device.roll(speed=0xFF, heading=0)
14  time.sleep(2)
15
16  # Move Sphero full speed heading 180 for 2sec
17  device.roll(speed=0xFF, heading=180)
18  time.sleep(2)
19
20  # Stop roll
21  device.stop()
22
23  device.disconnect()
```

Example 1 - Basic usage of the Sphero

Example 1 demonstrates the usage of a single Sphero device and the synchronous search functionality of the Sphero Manager. The device is discovered and returned as a Sphero Object by the Sphero manager (*line 7*). A connection must be established before any commands can be sent to the device (*line 10*). In line *13, 17, 21* the application sends movements commands to the connected device. The device is disconnected before the program terminates.

This example would have the Sphero drive in one direction for 2 second turn 180° and drive for another 2 second before ending in a full stop.

• **Asynchronous device discovery**

```
1  import time
2  import sphero
3
4  # list of found devices
5  devices = []
6
7
8  # callback when device is found
9  def on_new_sphero(device):
10     devices.append(device)
11
12 # Get sphero manager
13 sphero_manager = sphero.SpheroManager()
14
15 # Set callback
16 sphero_manager.set_sphero_found_cb(on_new_sphero)
17
18 # Start asynchronous search for devices
19 sphero_manager.start_auto_search()
20
21 time.sleep(60)
22
23 sphero_manager.stop_auto_search()
```

Example 2 - Asynchronous discovery of Sphero devices

Example 2 shows how to find multiple Sphero devices by using the asynchronous discovery support from the Sphero Manager.

A callback is registered (line 14) with the Sphero manager and this callback is triggered whenever a new device is discovered by the manager. The discovered devices is in this example are append to a list. The asynchronous Sphero discovery service runs in its own thread and the discovery service is started by the application in *line 10*.

This example would start the asynchronous discovery service and run a search for nearby Sphero's for one minute before terminating.

• Asynchronous data streaming

```
1   import time
2   import sphero
3
4   sphero_manager = sphero.SpheroManager()
5   device = sphero_manager.get_available_device()
6
7   device.connect()
8
9   def on_data(data):
10      # print Gyro angle x in degrees
11      print "GYRO", data.gyro.gyro_degrees.x
12
13  # set callback for data received from device
14  device.set_sensor_streaming_cb(on_data)
15
16  ### Create streaming config
17  ssc = sphero.SensorStreamingConfig()
18
19  # Set streaming speed 10Hz
20  ssc.sample_rate = 10
21
22  # Number of packets to stream
23  ssc.num_packets = ssc.STREAM_FOREVER
24
25  # Activate streaming of GYRO
26  ssc.stream_gyro(True)
27
28  # Start streaming with this config
29  device.set_data_streaming(ssc)
30
31  time.sleep(60)
32
33  device.disconnect()
```

Example 3 - Activate streaming from Sphero device

The streaming support in Sphero NAV is a feature that allows easy access to sensor data onboard the Sphero devices. Example 3 shows how to activate streaming from a Sphero device. Sensor streaming is an asynchronous feature where sensor data is given to the application in the form of a registered callback (line 13).

To configure the data to stream from the device a *SensorStreamingConfig objects* is used (SSC). The SCC holds settings for the data to stream, the frequency and the number of packets. In this example, streaming of the GYRO sensor is activated (*line 25*). When the Sphero object receives data from the device it triggers the registered callback with the sensor data passed as parameter. The data is given as a *SensorStreamingResponse object* (SSR). The SSR allow for easy access of the received data. In this example the angle of gyro x is retrieved in degrees and printed (*line 10*).

This example would connect to a device and print the Gyro x angle 10 times per second before terminating after 60 seconds.

- **Object tracker**

Example 4 shows how to use the tracker. Traceable object instances are created for two objects to track (*line 7, 8*). A filter for each objet is created and configured to find colors inside a blue and orange color range (*line 11* → *18*).

In this example, the color ranges are set in HSV (Hue, saturation, brightness/value) format. HSV is a more intuitive format of setting color ranges, and is the format normally used in tracking systems. The brightness value of HSV makes it easier for setting the color range to track. Note that the filter supports colors in other formats. (e.g. filter.lower.rgb = (0, 0, 5))

The objects to track are passed to the tracker in *line 39,* and the tracking is performed in an infinite loop. The position, heading and speed of the tracked object are printed for each iteration.

```
1   import tracker
2
3   # Create a tracker object
4   object_tracker = tracker.ColorTracker()
5
6   # create a traceable object
7   blue_traceable_object = tracker.TraceableObject()
8   orange_traceable_object = tracker.TraceableObject()
9
10  # Create a color filter for finding a blue object
11  blue_filter = tracker.ColorFilter()
12  blue_filter.lower.hsv = (100, 100, 100)
13  blue_filter.upper.hsv = (120, 255, 255)
14
15  # Create a color filter for finding a orange object
16  orange_filter = tracker.ColorFilter()
17  orange_filter.lower.hsv = (160, 113, 146)
18  orange_filter.upper.hsv = (180, 201, 255)
19
20  # Add filters to traceable objects
21  blue_traceable_object.filter = blue_filter
22  orange_traceable_object.filter = orange_filter
23
24  # Add objects to list
25  traceable_objects = [blue_traceable_object, orange_traceable_object]
26
27  # Run tracking on traceable objects
28  while True:
29      # Perform a tracking
30      traceable_objects = object_tracker.track_objects(traceable_objects)
31
32      # Iterate objects
33      for traceable in traceable_objects:
34
35          # Get data from the tracking if object was found
36          if traceable.last_tracking_successful:
37              print traceable.pos
38              print traceable.direction
39              print traceable.speed
40
```

Example 4 - Tracker code example

• Set and use events on PS3 controller

```python
import ps3

def button_callback():
    print "callback"

def axis_callback(value):
    print "axis", value

ps3_manager = ps3.PS3manager

controller_a = ps3_manager.get_available_controller()
controller_b = ps3_manager.get_available_controller()

### SET SINGLE EVENTS ###
# onButton press
controller_a.set_button_press_event(ps3.BUTTON_CIRCLE, button_callback)

# onButton release
controller_a.set_button_release_event(ps3.BUTTON_CIRCLE, button_callback)

# onButton axis
controller_a.set_axis_change_event(ps3.AXIS_CIRCLE, axis_callback)

### SET MULTIPLE EVENTS ###
controller_b.set_events(
    button_press={
        ps3.BUTTON_CROSS: button_callback
    },
    button_release={
        ps3.BUTTON_CROSS: button_callback
    },
    axis={
        ps3.AXIS_JOYSTICK_L_HOR: axis_callback
    }
)

# Start listening for events
ps3_manager.start()
```

Example 5 - Simple PS3 controller usage

Example 5 shows how to use the PS3 manager to retrieve a PS3 object. Events are bind to the controller in the form of callbacks. The PS3 object support multiple approaches for adding event callback. Events can be added one by one for the different event types (line 16, 19, 22) or added altogether (*line 25-34*).

Chapter 5 - Implementation

5.1 Introduction

This chapter goes into implementation details from the core functionality of Sphero NAV.

5.2 Technologies used

Python was used as the implementation language of Sphero NAV. Python is a productive language and for a system with a main purpose to be used in the recruitment program the author considers this is good choice. Developers that use the system would most likely have limited time to spare for new demos, and python allows for applications to be implemented rapidly. Python is also a much used and well known language at the University of Tromso.

Object tracking, video capture and visualization is implemented with support from Open CV. Open CV[28] (Open Source Computer Vision Library) is a computer vision and machine learning software library. The Open CV library has over 2500 implemented algorithms and tools for computer vision and machine learning. The algorithms can be used to track and identify objects, faces, movements any much more. Open CV has C++, C, Python, Java and MATLAB interfaces and it is supported on Windows, Linux, Android and Mac OS.

The Camera settings manager GUI was created using python Kivy[29]. Kivy is an open source Python framework that allows for easy implementation of functional graphical user interfaces.

[28] http://opencv.org/about.html (accessed 13.05.2014)
[29] http://kivy.org/#home (accessed 13.05.2014)

5.3 Object Tracking

5.3.1 Algorithm

The pseudo code in Figure 16 shows an overview of the tracking algorithm used by the tracker to locate objects.

```
1
2  def track_objects(traceable_objects):
3      # get current video frame
4      image = get_current_video_frame()
5
6      # Timestamp for this tracking
7      timestamp = get_timestamp()
8
9      # Repeat process for each traceable object
10     for traceable_object in traceable_objects:
11         # Get tracking filter
12         tracking_filter = traceable_object.get_filter()
13
14         # Create mask
15         raw_mask = tracking_filter.create_mask(image)
16
17         # Remove noise
18         mask = noise_reduction(raw_mask)
19
20         # Find all blobs in mask
21         blobs = find_blobs(mask)
22
23         # Find position of largest blob
24         pos = find_position_of_largest_blob(blobs)
25
26         # Save tracking data for this object
27         traceable_object.add_tracking(pos, timestamp)
28
29     return traceable_objects
```

Figure 16 - Shows pseudo code of the tracking algorithm

The tracker takes a list of traceable objects as input. Each object represents a device to track and must inherit the *TraceableObject class.* Tracking of objects is performed on the same image to ensure the correct position between each device relative to time. A timestamp set for each tracking allows users to access the capture time of the images used. Image capturing is performed with support from the OpenCV library.

Each traceable object holds a tracking filter. Filters contain logic for a binaryzation process used to create masks (see Figure 17). A mask is a two-dimensional Numpy[30] array containing black and white pixels. White pixels represent the areas (blobs) of the image that has passed the filter criteria. A perfectly configured filter should return a mask that holds one blob equal to the shape and position of the object it is supposed to mask out.

Sphero NAV allows developers to write custom filters. All filters must inherit the *BaseFilter* class. The filter is used by the tracker by passing an image to its *create_mask* method. Create mask analyzes the image and returns the mask that masks out the object.

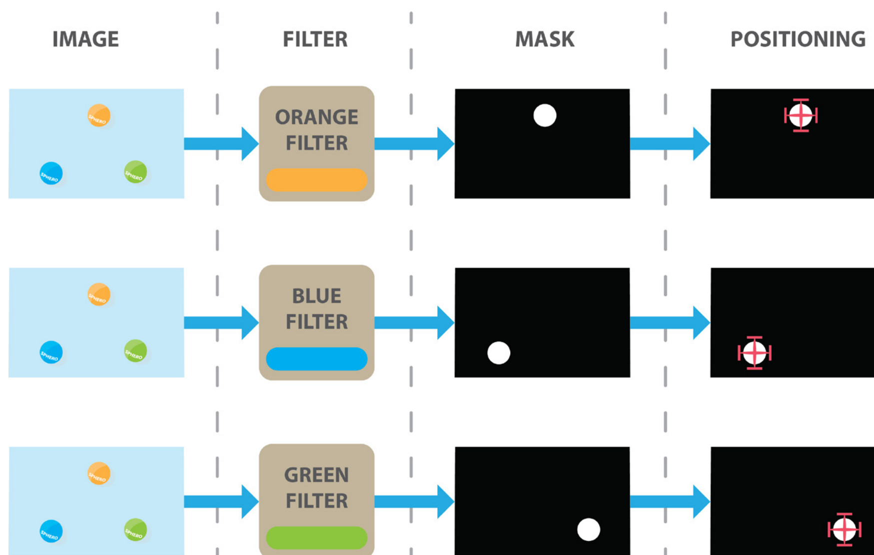


Figure 17 - image → filter → mask → position process

The filters currently used in Sphero NAV uses a color-based tracking. This approach finds pixels inside a given color-range (see Figure 17). When using color-based tracking, there is high chance of the mask containing noise. Noise appears when the captured image contains other objects or light that match the criteria of the filter (Figure 18). Noise in an image will create aditional

[30] http://www.numpy.org/ (accessed 30.05.2014)

blobs in the mask. A noise reduction step is used to limit the amount of blobs created from noise. A two-stepped approach where eroding and dilating (shrinks and grows) the edges of the blobs are used. This is performed with supported from OpenCV. Erode removes a specified size of the borders of every blob, blobs smaller than the eroded area are removed. Eroding will remove pixels from all blobs in the mask, including the blob that represents the device. Dilate is the reverse of eroding and is used to replaces the pixels that was removed from the blobs still in the mask.

Figure 18 - Filter containing noise

Despite the noise reduction step, there is most likely more than one blob still in the mask. A chain code algorithm[31] from OpenCV finds the position and size of all blobs remaining in the mask. The tracker assumes that the largest of these blob is the object it should track. The x, y coordinates of the center of the blob is set as the traced position. Positions used in Sphero NAV are based on the size of the image used for the tracking. See Figure 19 for the coordinate system Sphero NAV implements.

[31] http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_contours/py_contours_begin/py_contours_begin.html (accessed 31.05.2014)

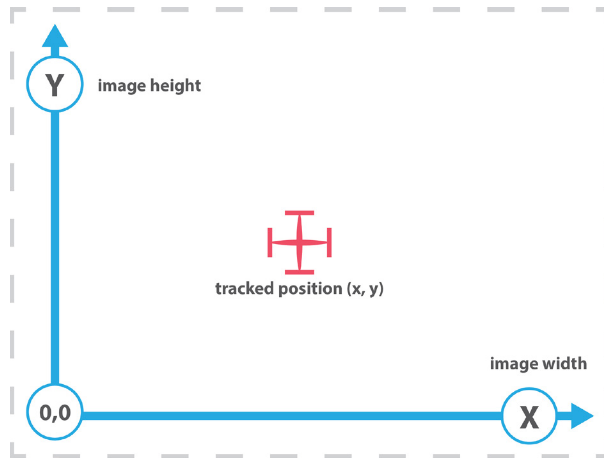


Figure 19 - Coordinate system used by the tracker

5.3.2 Traceable object and sample class

The TracableObject class (Figure 20) is the class that holds logic and stores data for trackings performed on an objects (e.g. Sphero). Every tracking of an object would result in a new sample added to a list of "tracking samples" held by the traceable object. For memory purposes, the users can specify the maximal number of samples to store.

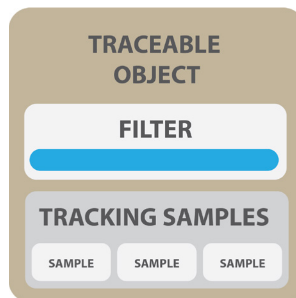


Figure 20 - Traceable object

A tracking sample stores the position, timestamp, and the state of the tracking (Successful or Non-Successful tracking). Every sample holds a reference to the sample prior to itself. This makes functionality that can calculate the angle, distance and speed between two samples easy. Calculations are performed with help from Vector2D objects created for Sphero NAV.

The vector2D class is a class that implements a representation of a 2D vector. This class holds different operations to use on the vector (e.g. normalize

magnitude, rotate, get angle ++). The Vector 2D class can be used by the application trough the utility library provided by Sphero NAV.

5.3.3 Filter

The filter class currently implemented in Sphero NAV are Color-filters. The filters are used for masking out pixels inside a given pixel range. The filter is implemented with support from OpenCV and Numpy. The filter holds two Color instances for its upper and lower range of pixels to mask out. An OpenCV method for finding pixels inside a given range is used to create the tracking mask.

The color class is a generic object that was created to allow easy usage of colors in Sphero NAV (e.g. graphics, filters ranges). The class allows for setting and getting of colors in different formats (e.g. RGB, HSV, HEX). Color objects can be used trough the utilitly library.

5.3.4 Camera controller

The camera controller was implemented by using the linux command line tool V4L-utils[32] (video for Linux). V4L allows control of web-camera setting trough the command line. The camera controller uses this functionality by sending system commands with pythons sub process library. The commands used for each web camera may differ and this restricts the current implementation of the camera controller to Logitech C920 web camera.

Even though the camera controller currently only supports the C920, it was designed to be configurable to other cameras as well. Each supported setting of a web camera is represented by V4L as properties (e.g. Exposure, auto focus, white balance). The settings of the connected camera are controlled by writing/reading of values for each property trough V4L. Other camera models could be used if its properties are mapped to a new camera controller object.

The graphical user interface (GUI) of the camera controller was implemented with support from Kivy. The GUI controls the camera settings by using the interface of the camera controller class.

[32] http://www.linuxtv.org/wiki/index.php/Main_Page (accessed 30.05.2014)

5.4 Sphero Module

5.4.1 Communication

All communication with the Sphero is performed over a Bluetooth connection. The Sphero object implements the interface where the application can send and receive data. Pybluez[33] was used on the client side as the library for handling Bluetooth communication. The Sphero object is designed so it is possible for multiple threads to communicate with the same Sphero device. This is useful for occasion where multiple threads perform different actions on the same device (e.g. movement, lights, vector controller, sensor readings). On top of this the Sphero object also supports asynchronous data. All communication is performed on the same RFCOOM[34] socket, address and port. RFCOOM is a Bluetooth protocol similar to TCP[35].

An asynchronous packet receiver (Figure 21) was created for handling both asynchronous and synchronous communication on the same socket. The receiver runs in its own thread and receives all incoming data from the device. The receiver is responsible for receiving incoming data, parse it into suited response objects or asynchronous packets and then notify the correct party.

When a thread sends a command by using the Sphero object it is blocked until the response for that message is received or timed out. A timeout will result in an exception passed to the application. All data received from the Sphero are parsed into response objects. There exist different types of response objects, and the request type sent to the device determines this. Simple python reflection is uses to parse the data into the correct response classes. A response class is an object that parses the raw response data and allows the application to access this data in aggregated or raw form.

[33] https://code.google.com/p/pybluez/ (accessed 25.05.2014)
[34] https://developer.bluetooth.org/TechnologyOverview/Pages/RFCOMM.aspx (accessed 25.04.2014)
[35] http://www.ietf.org/rfc/rfc793.txt (accessed 25.05.2014)

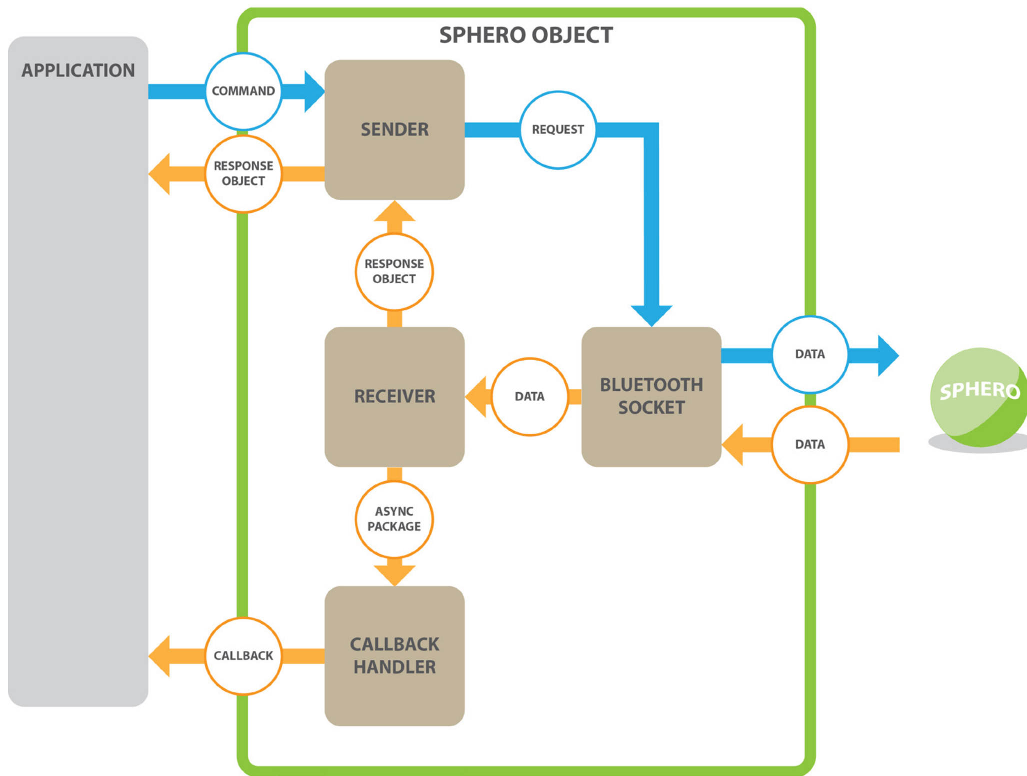


Figure 21 - Data flow Sphero Object

Asynchronous messages (e.g. sensor streaming, collision detection) are distributed back to the application in the form of pre-registered callbacks. The current implementation supports callbacks for sensor streaming, collision detection and power state notification. The Sphero object could easily be extended to support more of the functionality, but this was not prioritized in this project.

Parsing of data for all packets is handled with the use of python's struct library[36]. The library is used to parse binary data to and from python primitives.

The receiver runs in its own thread and reads incomming data from the Bluetooth socket. As discussed in the Sphero chapter, asynchronous and synchronous packets are sent in different formats. The receiver uses this structure to distinguish asynchronous and synchronous packets and it looks at

[36] https://docs.python.org/2/library/struct.html (accessed 25.05.2014)

the data field for knowing how much data to read from the socket for each packet.

5.4.2 Sphero Streaming

Allot of work was put into the sensor streaming support provided by Sphero NAV. A full streaming library that allows for configuration and streaming of sensors data from the Sphero device was implemented. The streaming support implements the Orbotix sensor Streaming API [7] [20].

A *Sensor Streaming Config* (SSC) object is used by the application to configure the sensor data to stream. Streaming of sensor data is activated on the Sphero with the synchronous command set_data_streaming(). The command normally takes two 32-bit strings as parameters. Each bit represents a data type to stream (e.g. Accelerometer, gyroscope, motor data). The SSC implements this setup in a more user-friendly approach. The SSC holds an interface containing methods for activating and deactivating sensors to stream. At the time where sensor streaming is activated by the application, the SSC is used to generate the raw bitmasks that is used in the request for activating streaming. This approach hides the low-level details from the developers and allows for easier activation of the streaming service.

Sensor data is received as asynchronous packages from the device. A Sensor streaming response class (SSR) is used to parse the sensor data. Raw data is parsed in the SSR by using the sensor-streaming configuration to check which sensors that are activated for streaming. The data is parsed into designated sensor classes held by the SSR. Each sensor class implements logic for formatting the raw sensor data. This allows the developer easy access to sensor data in various forms and hides the raw data received from the device.

5.4.3 Sphero Manager

The Sphero manager is implemented by using PyBluez Bluetooth discovery support. PyBluez allows for easy discovery of nearby device. A Bluetooth search would return a list of all discovered nearby devices in the form of Bluetooth addresses. Note that a search result would include previous discovered devices. To single out Sphero's from the search, a name lookup to each device is used. Devices that have a Bluetooth name that start with "Sphero-" are considered to be Orbotix Sphero's. Even though this has not given any problems so far, it is probably not the best approach. Other devices are allowed to set their own Bluetooth names, and if a device decided to use a name starting with "Sphero-*" it would be picked up by the Sphero manager as a Sphero device.

A problem noticed during development, was that some devices including the Sphero was terribly slow on name lookups. To gain some performance on each discovery iteration, a name cache was added. The names of previously discovered devices are stored in a hash map (python dictionary). This allows the Sphero manager to only lookup names for newly discovered devices. This approach gave some speed-up to the Bluetooth search. The application has access to flush the name cache if this should be desired.

Whenever a new Sphero device is discovered, the manager creates a Sphero Object that represents the that device. Sphero objects are held in a list in the manager. Sphero Objects stores the Bluetooth name and address of the device. The Bluetooth address is used later when the application wants to establish a connection to the Sphero device by using the connect method.

When the manager is used in asynchronous mode, the Bluetooth discovery service is ran in its own separate thread. For every Bluetooth discovery performed, the manager checks if any new Sphero devices is discovered, new devices are returned back to the application as Sphero Object using a callback registered by the application.

When the manager is used in synchronous mode, discovered devices are set as the return value of the search method.

5.4.4 Sphero Calibration

The movement of Sphero is controlled by using a Roll command from the Sphero Object. The roll command takes speed and heading as parameters. Sphero uses an *Internal Reference heading* to determine the real world direction that should be used for heading 0°. The heading passed to the Roll command is always relative to this. The Reference heading is set to 0° on Sphero startup. This means that the real world direction of the reference heading is the same as the angle of the Sphero core (IMU) on startup, and will differ from every usage.

The set heading command is used to reconfigure the real world direction used as the reference heading. The IMU is turned to the preferred direction, and the set heading command sets this angle as the new the reference heading for the Sphero to use.

To combine the vector controller and the positions from the tracker it is necessary to align the reference heading with the coordinate system used by the tracker. Aligned systems mean that when giving the Sphero device a roll command of 0° it should run in a straight line down the Y-axis of the tracked image. To achieve this, Sphero NAV supports a simple calibration algorithm to align the device and the tracker as best as possible. Figure 22 shows the six-stepped calibration algorithm implemented in the system.

Note: 0° in Figure 22 is drawn in Sphero coordinates and would equal 90° if it were drawn in Sphero NAV coordinates (Euclidian).

The six steps of Sphero calibration:

1. Turn the IMU to its current internal reference heading
2. Track the position of the device
3. Drive Sphero in a straight line with heading 0°
4. Track the position of the device
5. Calculate the tracked direction and tracked distance
6. (A) Turn the Sphero device equal degrees as the opposite of the tracked direction. The IMU has now the angle equal to 0° of the tracker system. (B) Set this angle as Sphero reference heading. The reference heading is now aligned with the trackers heading 0°.

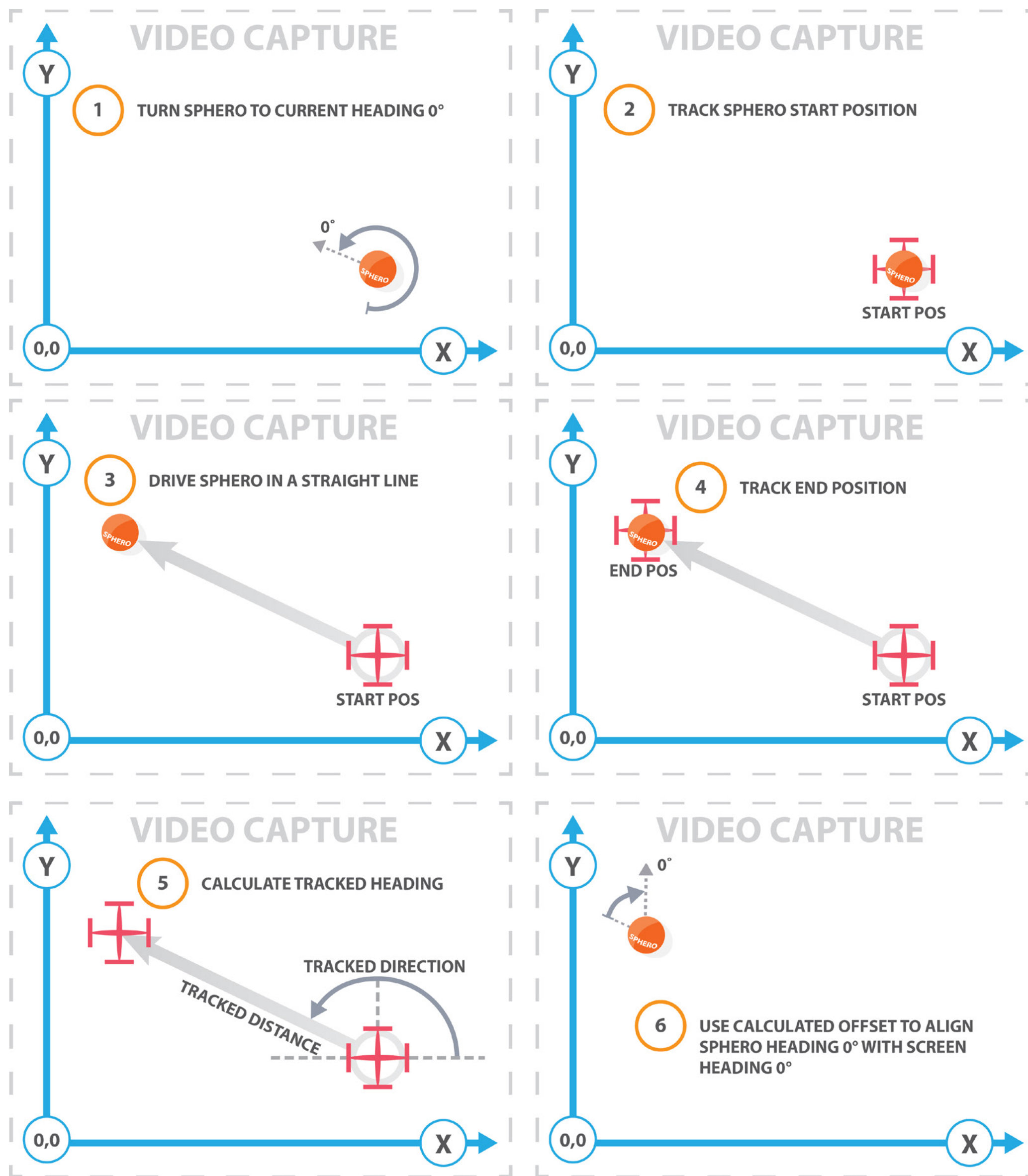


Figure 22 – Sphero calibration

The calibration support is implemented in the Traceable Sphero Object (4.3.2). It is the application that determines when to run a calibration, and this is triggered trough the calibrate method provided in the Traceable Sphero Object.

The six step calibration is the approach currently used in Sphero NAV. This approach was used to align the sensors and the internal locator system of the device discussed in the Sphero chapter (3.4.2) as well. Aligning of all system makes it easier for the developer to use sensor data and the internal locator data from the Sphero can be utilized if wanted by the application.

Another approach that could have been used that would not need a calibration step would be for the tracker to continually keep an internal offset between the Sphero and the tracker coordinate system. By adding this offset to Sphero's Roll commands the same result is achieved. The limitation is that the onboard systems on Sphero are not aligned and would also ned to be recalculatet for each usage.

5.5 PS3 Module

The PS3 module was implemented with support from Pygame[37]. Pygame is a python library that holds among many other things, support for connecting and using game controllers.

A PS3 object class was implemented as a wrapper for handling events from the game controllers. The PS3 class holds a list of registered callbacks that is mapped to the interface of the controller. This mapping is performed by the application.

The PS3 manager is used to discover connected controllers and handle events. Events are handled in an event loop running in its own thread. Whenever an event from a PS3 controller is captured, the manger will serve this event to the responding controller object. The controller object will parse the event and trigger the correct callback back to the application.

[37] http://www.pygame.org/news.html (accessed 31.05.2014)

Chapter 6 - Evaluation

6.1 Introduction

This chapter starts with by evaluating the Sphero NAV system. It discusses issues with the current design and solutions to improve the system.

6.2 Experiments

6.2.1 The experimental environment

The experiments was performed on a desktop machine running on an quad core Intel i7-3770 CPU with 3.4Ghz 16GB RAM and Linux mint Petra running Native. Bluetooth was performed over a Star Tech USB Bluetooth dongle. Up to three Orbotix Sphero devices where used in the experiments. A Logitech C920 web camera was used for tracking.

6.2.2 Communication

Sphero devices are operated by commands sent from the client. It is therefore crucial that the round trip time (RTT) and update rate of the commands is good enough so applications can control the devices without any problems. A measure of the RTT from the client to the Sphero was performed to benchmark the performance of the communication channel to and from a device.

Each test was performed with 1 000 iterations and the average time of these samples was used as the result. Two tests where used for testing the RTT, both tests performed a Ping command [7] to the device. A ping command sends a request to the device and a result package is sent back to the client. The packets sent was 7 Bytes in size. The time of each ping was used as a measure for the RTT of one command. The first test was performed with data streaming disabled on the device. The second was performed where data from all

sensors was streamed back from the device asynchrony with a velocity of 20 packets per second (20Hz). Each packet streamed from the device was 67 Bytes in size. Figure 23 shows the average RTT of the tests performed.

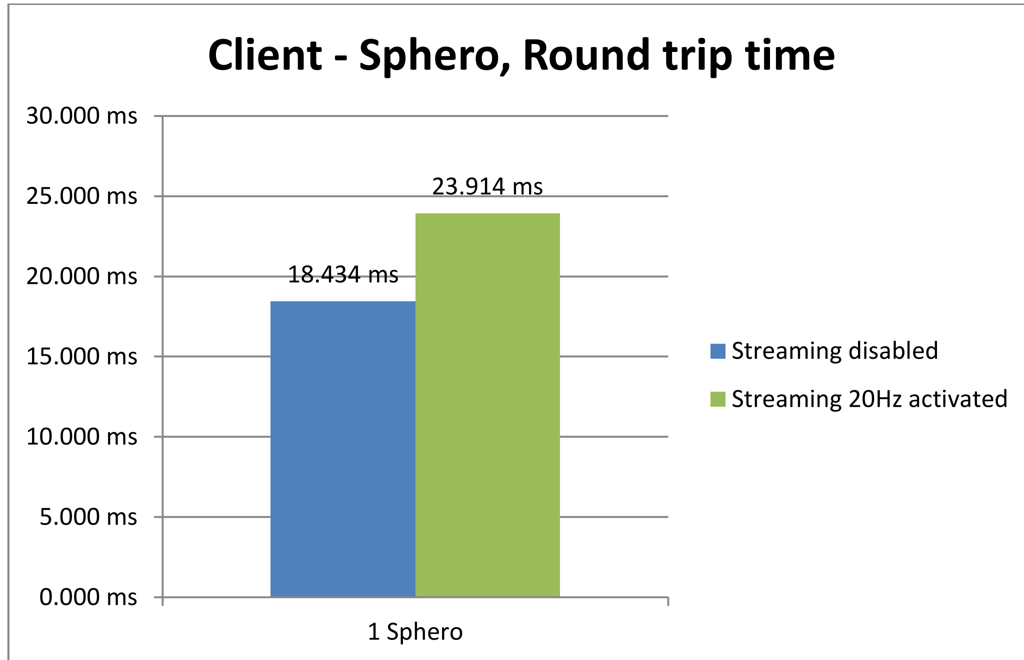


Figure 23 - Client - Sphero RRT

The result from the RTT measures shows that it is possible to send ~55 packets per second when streaming is disabled and ~43 packets when streaming is activated. This update rate is more than enough in both cases to operate the Sphero device without any problems.

The Sphero allows for streaming of sensor data. Application can use this data and the streaming speed is configurable by the application. In most cases, it is desired to get this data in a fixed interval, this allows for the update rate to be used in calculations that uses the sensor data. It is therefore necessary that the update rate is as stable as possible.

A test for measuring the performance of the streaming service was performed. The test benchmarked the max streaming speed the Sphero NAV system could achieve and still keep a stable update rate to the application. The test was performed 3 times with one, two and three connected devices. Each test was performed with eight different streaming speed activated on the Sphero's.

Each iteration measured the time it took to receive 10 000 asynchronous streaming packets from each device. The total time used for each device was divided by the number of packets received. This result was used to calculate the transfer rate in Hz. Figure 24 shows the results of the test.

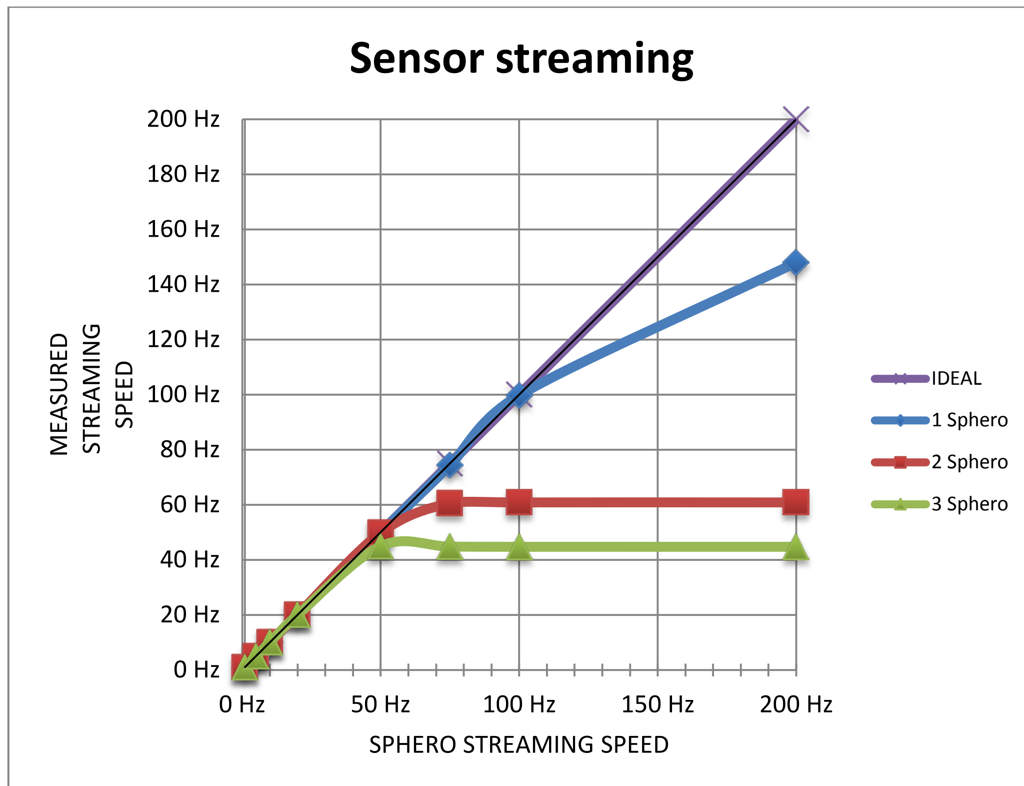


Figure 24 - Graph of streaming speed

The achieved streaming speed indicates that Sphero NAV handles stable streaming speeds for three devices up to approximately 50Hz. Orbotix says in their API that a streaming speed of 20Hz is enough for getting accurate sensor information in most cases. These result shows that the streaming service has good enough performance to be used for applications that want to take use of sensor data.

Note: The testing environment used for these test was polluted with many different Bluetooth devices and Wi-Fi networks. This could have affected the result of the measurements.

6.2.3 Tracking performance

The applications that would use position data from the tracking of devices would in almost all cases want updates of positions as fast as possible. The performance of the tracker was to measure how many frames/tracking's per second (FPS) it could achieve with different settings. The tests were run with four different image resolutions. Each test was tested with 0 to 3 devices to track. Each configuration was run for ~1000 iteration, and the average FPS of each test was used as the result. The CPU load for each test was tracked by using the LINUX command line tool TOP[38]. The achieved FPS of each configuration is displayed in Figure 25. The CPU load is displayed in Figure 26.

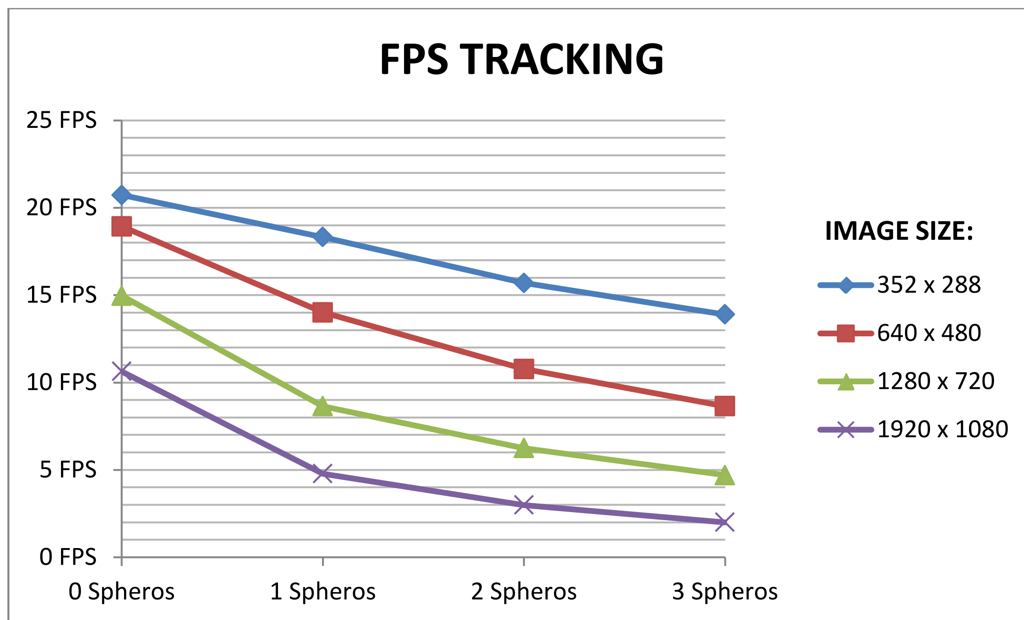


Figure 25 – Graph of FPS Tracking

[38] http://linux.about.com/od/commands/l/blcmdl1_top.htm (accessed 31.05.2014)

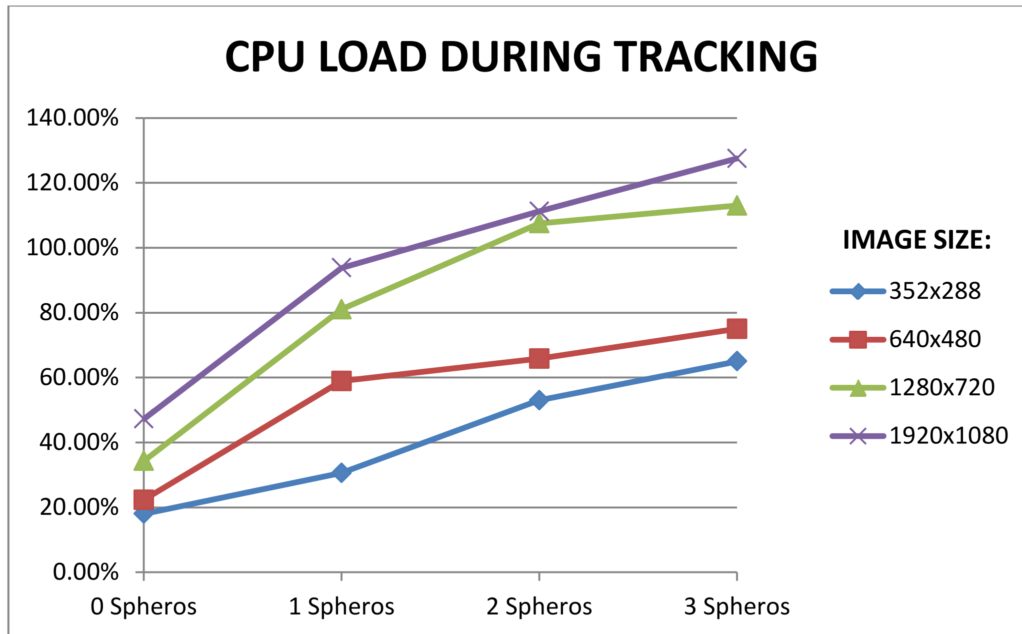


Figure 26 - Graph of CPU tracking

The results from the performance tracking shows that increasing the number of objects to track and/or the size of the images captured by the camera decreases the performance of the tracker rapidly.

The frame size used during development and testing of the system is 640x480. With three devices and this image resolution, the system achieves approximately 10 FPS. This is a sufficient update rate of the devices position to use in application, but higher FPS is of course wanted.

6.2.4 Library test

To test each component of the Sphero NAV library a test application was created. The application used every component provided from the library and was used to check if the library worked as intended.

The application used two devices. The tracker tracked each device successfully and the retrieved position was used in the application for adding different functionality.

Each Sphero devices was controlled by using the PS3 module. Axis events where used together with the vector controller to test manual steering of the Sphero's. Different functionality from the Sphero API where mapped to the PS3 interface (e.g. Light, ping, data reading). This worked as intended and both Sphero's was successfully controlled simultaneously with its own PS3 game controller.

The application implemented a simple demonstration of adding virtual borders. The Sphero's drove around autonomously in a bouncing ball like fashion. When a device was tracked near one of the edges, the heading of the Sphero was changed to the center of the image. This functionality worked successfully, but the virtual borders had to be placed within a good range of the edges of the image for having the devices turn before they were outside the tracking area.

Virtual point functionality was created for testing semi-autonomous usage. The virtual point was a point in the tracking image that the Sphero it belonged to where drawn towards (see Figure 27). The point was moved manually by using a PS3 controller. The heading of the Sphero was always directed toward this point, and the roll speed was determined by the distance from the virtual point to the Sphero. This approach of controlling the devices was very successful and proved itself as an easy way to control the Sphero. The author believes that this way of controlling the devices is a good approach an could be used in other applications as well.

Another test involving virtual points was to control on Sphero manually and have the other follow the first one. The position of a virtual dot used by the second Sphero was always updated to position of the manual controlled

Sphero. This created a slave like behavior where the second Sphero was always drawn toward the manual controlled device.

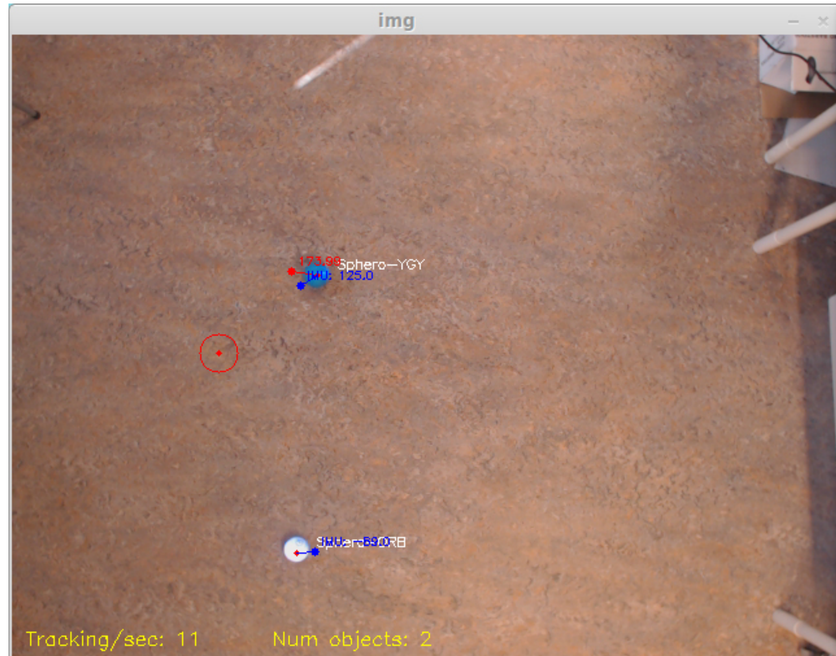


Figure 27 - Follow virtual dot test

The test application demonstrates that Sphero NAV can be used to create different types of application and that it is a usable library.

6.2.5 Video

A video of the functionality described in the previous section and a demonstration of the Sphero NAV system can be found at the following site [8]. The video demonstrates that the Sphero NAV system works as inteneded.

6.3 Known bugs and issues

6.3.1 Spikes in communication

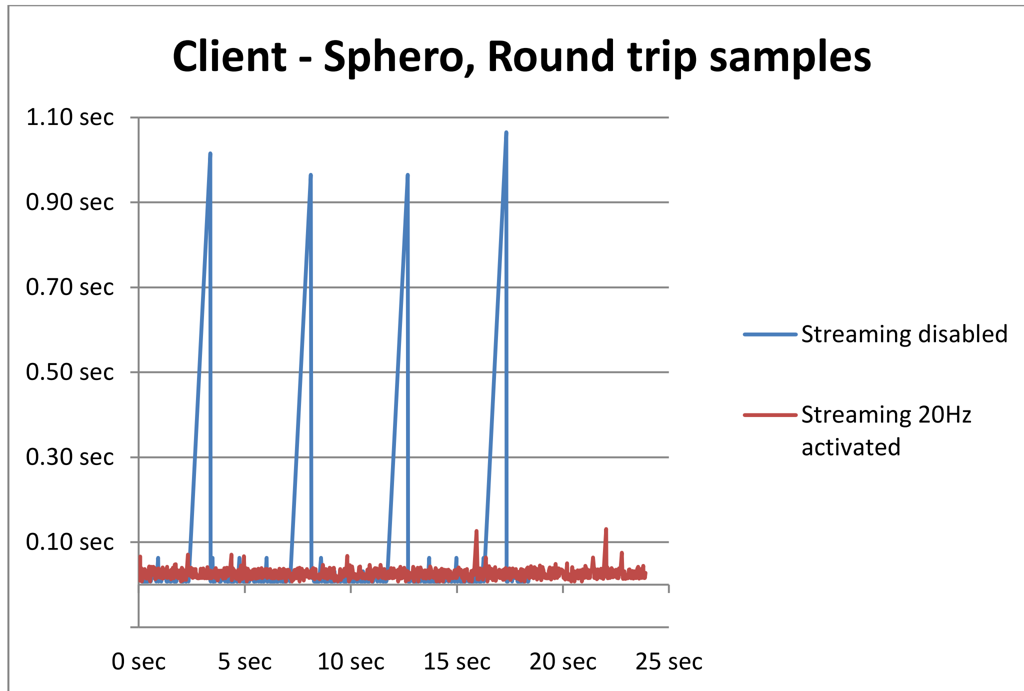


Figure 28 - Plotting of round trip samples

During the performance testing of the communication, a strange bug was discovered. Figure 28 shows all the samples used in the test for measuring the RRT from the client to the Sphero. As shown, some strange spikes occur in a fixed interval. The spikes are longer when streaming is disabled, but can be noticed when streaming is activated as well. The spikes appeared on multiple reruns of the experiment. Unfortunately, there was no time left before the deadline of this thesis to look into this result. However, one thing noticed is that this happens for every 255 packet sent to the device. The author believes that the results has something to do with the sequence number used in the packages sent to the device. The range of sequence numbers used by the Sphero API packets goes from 0 - 255. The number is increased for every packet by the client and wraps around to 0 when all number are used. One bug discovered in the client was the frequency number wrapped around on

244. There was not enough time to run a new test, but this issue could have created a hiccup on the Sphero.

6.3.2 Microsoft Kinect

One of the first ides was to use the same tracking approach as used in Open Pool 2.4. Open pool uses the depth camera from Microsoft Kinect for tracking the position of the billiard balls. This approach was first used in Sphero NAV, but was discarded for various reasons. The Kinect was difficult to use with python and needed drivers that where demanding to install and get to work correctly. When the Kinect was up and running it was successfully used for tracking of object on short distances, but when it was placed in the desired height above the tracking area, it was almost impossible to distinguish the balls from floor by using the depth data. Sphero NAV is therefore based on a pure image based approach.

6.3.3 Color tracking

There exist many approaches for object tracking. Sphero NAV currently uses Color tracking to distinguish objects. A limitation with color tracking is that it only allows for tracking of one object in the same color. Color tracking is also highly sensitive for getting noise from other objects and ambient light.

Other approaches was tested during development. Using a Kinect depth camera worked good on short distances but could not be used for tracking of Sphero's when the distance was too great. An algorithm for finding circular objects was also tested; this approach was unstable and limited the tracking to only finding Sphero's, and it was impossible to distinguish different devices. Last, a strobe tracking system was tested. The Sphero's was tracked in in turn by separate images. The Sphero that should be tracked in an image was glowing and the others would turn their lights off. This approached worked well and it was easy to locate the Sphero's. The problem was that it was terribly slow.

6.3.4 Internal reference heading

An issue discovered during the development of the system demonstrated that the Sphero devices internal set reference heading gets inaccurate after a couple of minutes of driving. This means that after ~3 - 5min of drive time the device's reference heading is so of from its calibrated reference heading, and needs to be recalibrated. This issue lies within the Sphero and will probably be

improved in future firmware versions for the device. The solution as of today is to rerun the calibration periodically.

6.4 Improvements

6.4.1 Support all platforms

The current version of Sphero NAV is only tested in Linux. All libraries used in Sphero NAV are supported in Windows and OS X and with minor modification, the system should be able to be ported to these platforms as well.

6.4.2 Bluetooth lookup is slow

The current version of the Sphero manager uses a slow approach for finding new devices. The asynchronous Bluetooth-discovery service used in the current version is implemented by first looking for new devices for a given period of time (typically 10sec). The application is not informed of new devices until this search is finished. This means that if the Bluetooth-discovery last for 10 seconds for each search, it would take at least 10 seconds before the first Sphero device is discovered.

The author has found a better approach[39], but had not enough time to implement this improvement. This approach support intermediate callbacks when new devices are found. By improving the Bluetooth-discovery system in the Sphero manager, the application could be notified instantly when a new device is discovered. This would decrease the discovery time for new devices.

6.4.3 Distributed system

One of the issues that the evaluation of Sphero NAV indicates is that the performance of the tracker decreases rapidly when tracing multiple Sphero's. Python was probably not the best language to use for Sphero NAV. The issue lies with the problem that when adding more devices there will be more threads to execute and more pixels to evaluate. Python's interpreter has currently no support for running threads in true parallel, thus limiting performance that could have been gained on multiprocessing machines [25] [26].

[39] http://people.csail.mit.edu/albert/bluez-intro/x339.html#pbz-adv-async (accessed 28.05.2014)

The author sees in retrospect that the system should have been implemented as a distributed system where each module was implemented as separate processes. The modules could have been connected through a RESTful [27] web interface and data could be transferred between the components in the form of JSON[40]. This approach would be more suited and make it possible to use different programming platforms for each component. This design would give additional overhead time due to communication between the different modules, but the author believes that the speed-up gained by using more suited technologies for each function would be higher.

A distributed system would also allow for a separation of functionality to several units. One approach could be to control Sphero devices from the end-users smart phones, and the position and coordination of the devices would be obtained by a tracking service running on a separate machine.

6.4.4 Image evaluation

The current implementation of the tracking system iterates over every pixel in each captured image, this happens one time for every object traced. This gives high overhead times when tracing multiple objects, and it decreases the number of frames tracked per second rapidly.

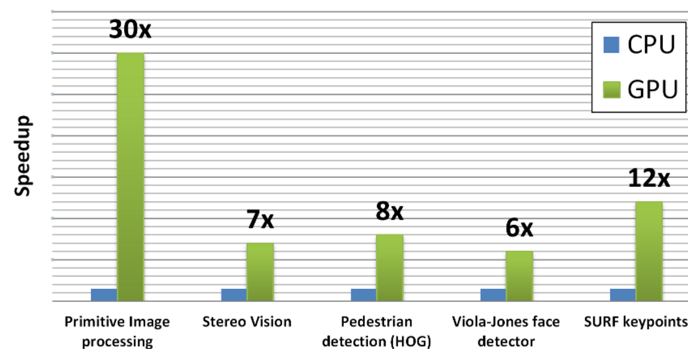


Figure 29 - GPU vs CPU processing (figure from[41])

A solution for speedup could be to combine all of the tracking filters used in one tracking to a combined filter. The combined filter is used to subtract the matching pixel areas of the captured image. Each area could then be split up

[40] http://www.json.org/ (accessed 01.06.2014)
[41] http://opencv.org/platforms/cuda.html (accessed 25.05.2014)

and identified separately to match each object that should be traced. This approach would minimize the number of pixels that would need to be iterated for each object, and could give some additional speed-up of the tracking system.

To really improve the performance of the tracking system, image processing should have been implemented with CUDA support [28]. CUDA[42] allows for offloading of computations to the Graphical processing Unit[43] (GPU). A GPU is designed to perform calculations on images in parallel. The tracking algorithm used in the tracker evaluates pixels independently making it an embarrassingly parallel problem[44]. Performing the calculations on a GPU would speed up the tracking significantly[45]. Figure 29 shows the massive speed up gained by using the GPU compared to a CPU based approach.

Open CV holds C and C++ support for taking use of the GPU in its calculations. This is something that should have been looked into to increase the performance of the tracker. There was not enough time to do this.

6.4.5 Improved Tracking

During the development phase of Sphero NAV, unforeseen issues with external devices and Bluetooth connections took up much time of the project. The python Sphero API took much longer time to get up and running than expected. Much of the code was rewritten for at all be able to connect to the Sphero device. This gave less time for improvements and optimization of the tracking part of the system.

The current implementation of Sphero NAV tracks devices successfully, but there are still allot of potential modifications that could be implemented.

There should be an easier way for the developers to create filters for tracking. An approach found in another system that uses object tracking allows for the user to interactively click on the object to track in the video capture[46]. A system like this could be extended into the tracking module, and the end-users

[42] http://www.nvidia.com/object/cuda_home_new.html (accessed 25.05.2014)
[43] http://www.nvidia.com/object/what-is-gpu-computing.html (accessed 25.05.2014)
[44] http://www.cs.nthu.edu.tw/~ychung/slides/para_programming/slides3.pdf (accessed 25.05.2014)
[45] http://www.youtube.com/watch?v=-P28LKWTzrI#t=56 (accessed 25.05.2014)
[46] http://www.virtual-drums.com/frozen-cameleon.php (accessed 20.05.2014)

of the system could just click on the desired Sphero to track in the captured image, and a filter for that object would be created.

The current version of the video capture requires that the users of the system manually adjust the settings of the image capture. This is done through the camera settings manager. An extension of the tracker that could perform auto adjustments of the video stream would be a useful feature, and would make the system easier to use for the developer.

Open CV implements good support for camera calibration[47]. This functionality should have been added to remove distortion in the images used for tracking. This would give a more precise result of the tracked positions. Radial distortion will typically appear in images from an uncelebrated camera. Radial distortion would make straight lines in an image appear curved; especially at the corner of the images. Radial distortion will affect the position of the tracked devices in some degree.

There exist many approaches for objects tracking that have not been tested in this thesis. The tracking system itself is a field big enough to be written as its own project. The remainder of this section gives some examples of other approaches that could have been used to improve the tracking of objects in Sphero NAV.

Background subtraction

Sphero uses a stationary camera to capture images. Background subtraction[48] (BS) is an approach that is used to remove the background of a tracking area. An image where only the background of the tracking area is present is captured. This image is subtracted from all other images removing the background. After a successful background subtraction, the tracked objects should be the only objects present in the picture.

[47] http://docs.opencv.org/trunk/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html (accessed 28.05.2014)

[48] http://docs.opencv.org/trunk/doc/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html (accessed 01.06.2014)

Template matching

Template matching[49] is an approach that uses template images to locate objects. The template of the object to track is used by the tracking system to look for objects similar to the template. The tracker could use templates of Sphero's glowing in different colors for tracking multiple devices.

Machine learning

TLD (Tracking-Learning-Detection) is a real-time algorithm for tracking unknown objects in a live video stream [29]. TLD uses Machine learning to locate objects. The author has not used much time for researching this approach, but it seems that this is something that could be used in the tracking system of Sphero NAV.

6.5 Problem definition solved

From the problem definition of this thesis presented in the introduction chapter (1.2):

> *"Develop a navigation platform for one or more users to control one or more robots (drones, sensor etc.). The platform should be easy to use and has to allow robots to operate on different levels of autonomy. The platform should also be easy to deploy and use both in the lab and when visiting schools and recruitment fairs."*

The remainder of this chapter explains how Sphero NAV has solved the problem definition of this project.

> *"Develop a navigation platform for one or more users to control one or more robots (drones, sensor etc.)."*

Sphero NAV implements a library that allows for tracking and control of one or multiple Sphero devices. The current version of Sphero NAV is restricted to tracking of the Sphero. However, its design and architecture allows developers to extend its functionality by writing custom filters and traceable object that allows for tracking of other types of devices.

[49] http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html (accessed 01.06.2014)

> *“The platform should be easy to use and has to allow robots to operate on different levels of autonomy”*

There has not been time to get feedback from other developers to test Sphero NAV’s ease of use. However, the author believes with background in the test application, and the code examples demonstrated that the Sphero NAV library is easy for developers with Python experience to use for new applications.

Sphero NAV has demonstrated trough the test application that the library can be used to control Sphero devices in a fully autonomous, semi-autonomous and manual mode. This demonstration proves that the Sphero NAV library allows the devices to be controlled on different levels of autonomy.

> *“The platform should also be easy to deploy and use both in the lab and when visiting schools and recruitment fairs.”*

Sphero NAV was implemented with Python. Python is a language that requires little work from the developer to get up and running. The Sphero NAV system implements a camera controller that allows its users to control and adjust the setting of the video stream. This support was created with respect for making the system dynamic and possible to use in different setting.

One of the limitations with the system may be that it is required to mount a camera in the celling over the tracking area. This can be difficult in some venues. However, if the University could invest in some sort of camera rig this should be a quick fix.

Based on the evaluation and testing of the Sphero NAV system the author believes that the Sphero NAV system has solved the problem definition and allows developers to create richer demonstration for recruitment purposes.

Chapter 7 - Conclusion

7.1 Conclusion

This thesis has outlined Sphero NAV – a software library for creating richer Sphero applications for use in recruitment. Sphero NAV provides functionality for tracking and controlling of multiple Sphero devices. Control of devices is possible either through pre-programed control systems or with the use of PS3 controllers.

Sphero NAV was evaluated and it was concluded that the project had solved the problem definition of this thesis. The evaluation of Sphero NAV demonstrated that it is possible to use Sphero NAV to create applications that uses tracker data to add extended functionality for demonstrations of the Orbotix Sphero.

In the end, the author is satisfied with the results of the project given the limited timeframe of this thesis.

7.2 Concluding remarks

One of the big issues in the development phase of Sphero NAV was to work with external devices. Countless hours have been used for establishing connections, debugging and using the devices. Programing with external devices really tests one's patience.

Creating programs for physical devices that moves in the real world is not the same as programming a game where all the physics are hard coded and can be controlled by the developer.

Real devices moves differently and the movement is affected by real world physics. A bouncing ball created in graphics visualization can change direction immediately; the direction of a driving Sphero on the other hand would need a good amount of breaking distance before it can turn. These types of calculations have to be taken into account when creating visualization or games that take uses of physical devices.

A bouncing ball in graphics visualization is programed to move in a straight line and is not affected by its underlying surface. A physical device on the other hand is affected by its surface and this makes the devices wobble from side to side changing its heading and position randomly.

7.3 Future work and ideas

Although Sphero NAV solves the problem definition there are still many opportunities for future work and improvements.

Some of the related systems outlined in this thesis have similarities to Sphero NAV in the approach they use for tracking the devices. The main difference is that the other systems uses Infrared (IR) tracking to find the devices. IR tracking is a good approach because it does not limit the physical appearance of device in the same way as color based tracking would. IR is invisible to the human eye and the tracking is not polluted from other objects in the same level as with color tracking. The problem is that the Sphero device does not currently hold any IR LED's, and this restricts this approach for Sphero NAV. It may be that Orbotix decides to add this to future versions of the Sphero. If they do, this should be tested as a new tracking approach for Sphero NAV.

Another feature used by the related systems is the use of a projector to display graphics on the floor where the devices are used. This is a feature the author dreamed of implementing, but had to drop because there simply was no time for development and money for equipment. The author hope that this is one of the things that could be added in the future.

The Sphero devices have by default the possibility to stream all of its sensor data to its connected client. A graphical user interface displaying this data would make it easier for developers to know what goes on inside the device. The Kivy framework could be used to create a Sphero widget that could display all onboard data from the Sphero.

Last as mentioned earlier there was not enough time to implement the tracking module of Sphero NAV to the level desired by the author. The tracking module has a high potential for optimization and extra features and better tracking algorithms should be added.

Chapter 8 - References

[1] R. Godman, A. Eguchi and E. Sklar, "Using educational robotics to engage inner-city students with technology," in *Depth of computer science*, Universiy of cambridge, Cambridge CB2 1QA, uk, 2003, pp. 214-221.

[2] E. D. Oppliger, "University - pre college interaction through first robotics competition," in *Session 6D3 - International Conference on Engineering Education*, 2001, pp. 11-16.

[3] R. Mitchell, K. Warwick, N. W. Browne, N. M. Gasson and J. Wyatt, "Engaging Robots: Innovative Outreach for Attracting," *IEEE TRANSACTIONS ON EDUCATION,* vol. 1, no. 53, pp. 105-113, 2010.

[4] J. Alonso-Mora, A. Breitenmoser, M. Rufli, S. Haag, G. Caprari, R. Siegwart and P. Beardsley, *DisplaySwarm: A robot swarm displaying images,* IROS 2011 open research demonstration, 2011.

[5] Orbotix, "Sphero Home page," [Online]. Available: http://www.gosphero.com/sphero-2-0/. [Accessed 20 4 2014].

[6] Python, "Python Programming language," [Online]. Available: https://www.python.org/. [Accessed 20 4 2014].

[7] Orbotix, "Sphero API 1.5 Documentation," 20 8 2013. [Online]. Available: https://github.com/orbotix/DeveloperResources/blob/master/docs/Sp

hero_API_1.50.pdf. [Accessed 20 4 2014].

[8] S. Nistad, "Sphero NAV demo VIDEO," Youtube.com, 29 5 2014. [Online]. Available: https://www.youtube.com/watch?v=KlWZrcMtZzI. [Accessed 31 5 31].

[9] K. Sachiko, S. Toshiki and K. Hideki, "Smart Ball and a NewDynamic Form of Entertainment," in *Playful User interfaces, Gaming Media and Social Effects*, Springer Science, Businnes Media Singapore, 2014, pp. 141-160.

[10] D. Holman and R. Vertegal, "Organic user interfaces: Designing computers in any way, shape, or form," *Communication of the ACM,* vol. 2008, no. 6 - vol. 51, pp. 48 - 55, 2008.

[11] C. Wisneski, J. Orbanes and I. Hiroshi, *PingPongPlus: Augmentation and transformation of athletic interpersonal interaction,* MIT Media Laboratory: MIT, 1998.

[12] J. Alonso-Mora, A. Breitenmoser, P. Beardsley and R. Siegwart, *Reciprocal collision avoidance for multiple car-like robots.*

[13] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley and R. Siegwart, *Optimal reciprocal collision avoidance for multiple non-holonomic robots,* Disney research zurich.

[14] J. Alonso-Mora, A. Breitenmoser, M. Rufli, R. Siegwart and P. Beardsley, *Multi-robot system for artistic pattern formation.*

[15] Orbotix, "Sphero whitelist process," 10 12 2012. [Online]. Available: https://github.com/orbotix/DeveloperResources/blob/master/docs/Sphero_Whitelist_Process.pdf. [Accessed 20 4 2014].

[16] K. Hall-Geisler, "How Sphero works," [Online]. Available: http://electronics.howstuffworks.com/sphero.htm. [Accessed 20 4 2014].

[17] Orbotix, "Sphero developer site," Orbotix, [Online]. Available: https://developer.gosphero.com/. [Accessed 18 5 2014].

[18] Faulkner, "Github, Python Sphero API," Faulkner, 2012. [Online]. Available: https://github.com/faulkner/sphero. [Accessed 20 4 2014].

[19] H. S. Oluwatosin, "Client-Server Model," *IOSR Journal of Computer Engineering,* no. 16, pp. 57-71, 2014.

[20] Orbotix, "Sphero locator documentation," 29 8 2012. [Online]. Available: https://github.com/orbotix/DeveloperResources/blob/master/docs/Sphero%20Locator%201.2.pdf. [Accessed 20 4 2014].

[21] Orbotix, "Sphero OrbBasic Interpreter," 06 5 2013. [Online]. Available: https://github.com/orbotix/DeveloperResources/blob/master/docs/orbBasic_1.07.pdf. [Accessed 20 4 2014].

[22] Orbotix, "Sphero Macros documentation v0.99," 8 4 2013. [Online]. Available: https://github.com/orbotix/DeveloperResources/blob/master/docs/Sphero_Macros_0.99.pdf. [Accessed 20 4 2014].

[23] Orbotix, "Sphero collision detection," 25 2 2013. [Online]. Available: https://github.com/orbotix/DeveloperResources/blob/master/docs/Collision%20detection%201.2.pdf. [Accessed 20 4 2014].

[24] R. Tougher, "Creating reusable software libraries," *Linux gazette,* August 2002.

[25] D. Beazley, "Understanding the Python GIL," 2010. [Online]. Available: http://dabeaz.com/python/UnderstandingGIL.pdf. [Accessed 18 5 2014].

[26] N. Matloff and H. Francis, *utorial on Threads Programming with Python,* University of California, Davis: University of California, Davis, 2007.

[27] T. R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures (chapter 5),* University of California, Irvine, 2000.

[28] J. Huang, S. P. Ponce, S. I. Park, Y. Cao and F. Quek, *GPU-Accelerated Computation for Robust Motion Tracking Using the CUDA Framework,* Center of Human Computer Interaction: Virginia Polytechnic Institute and State University.

[29] Z. Kalal, M. Krystian and M. Jiri, "Tracking-Learning-Detection," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE,* vol. 2012, no. 7, pp. 1409-1422, 7 2012.