Coventry University

Faculty of Engineering, Environment & Computing

Department of Mechanical, Aerospace & Automotive Engineering

324MAE Project Report

Development of Technological Platform for Personal Robotic Assistant with Autonomous Person-Following Capabilities

Submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering

M. Olak SID: 7179112

BEng Mechanical Engineering

Supervisor: Mr J. Pickering

May, 2020

Declaration: The work described in this report is the result of my own investigations. All sections of the text and results that have been obtained from other work are fully referenced. I understand that cheating and plagiarism constitute a breach of University Regulations and will be dealt with accordingly.	
Signed: Olak Michal	Date: 1/05/2020

Abstract

This report outlines the process of designing and making a technological platform for a personal assistant robot. The platform is designed to imitate human-like footprint, form factor and perception abilities by being based on a two-wheeled self-balancing robot and having autonomous navigation capabilities. The process of making the two-wheeled self-balancing robot is described in detail, from literature review, methodology creation and presentation of results. The robot is controlled using Proportional-Integral-Derivative controller and actuated using a set of high power DC motors driven by high precision electronic speed controller and powered by lithium-ion battery. This system setup is allowing the robot to be dynamically balanced within a limited oscillatory range. The autonomous navigation is informed by computer vision system based on the mobile computing platform and utilising deep neural network object detector. The work presented in this study is successfully achieving balancing control of the robot though PID controller and the velocity control by remotely setting the angle setpoint using radio system. Due to time constraints the

The increasing pace and difficulty of everyday tasks, and the ageing population with an increased need for personal care combined with a common presence of Internet of Things environments both create the need and allow for the robotic personal assistants to become a tangible reality.

The question of form and functionality of such a device is of a great depth and its exploration could be a project on its own. However, it could be generally stated that in order for such a device to operate efficiently in human environments, it should be created in the image and likeness of a human. Thus, the direction taken here is of creating a robot that utilises human-like shape factor and footprint, and is equipped in perception methods of similar characteristics.

Table of Contents

List of Figures

Nomenclature

x - Cart position [m]

$\dot{x}$ - Cart velocity [m/s]

θ - Pendulum angular position [rad]

$\dot{\theta}$ - Pendulum angular velocity [rad/s]

M_c - Cart mass [kg]

m_p - Pendulum mass [kg]

l_p - Distance from centre of mass to axis of rotation [m]

u - Input force [N]

g - Gravitational acceleration [m/s^2]

K_p

K_i

K_d

$u(t)$

$e(t)$

$T\ s$

$y(t)$

Abbreviations

TWSB – two-wheeled, self-balancing

IMU – inertial measurement unit

RC - radio control

CV – computer vision

DNN – deep neural network

ANN – artificial neural network

ESC – electronic speed controller

PID – proportional, integral, derivative controller

1. Introduction

1.1. Robotic assistant

The increasing pace and difficulty of everyday tasks, and the ageing population with an increased need for personal care combined with a common presence of Internet of Things environments both create the need and allow for the robotic personal assistants to become a tangible reality. Similarly to how the industrial automation has taken away the toll of performing the dull, repetitive tasks away from people and let the masses to perform more creative tasks and chase meaningful endeavours, the same way our everyday lives can be enhanced by automation of unwanted parts. The vision here is of a personal robotic assistant that could perform simple tasks that we don't want to or cannot do. Be it making a hot beverage, bringing something from the room next door or carrying luggage. Personal assistants already exist, as a common example of Siri or Alexa, however they lack the physical, tangible component. The physical hardware is the missing key and the stepping stone for automation of everyday tasks to become feasible reality.

The mission of developing such a device is a long and expensive journey. What is develop here instead is a technological platform that would serve as a backbone for further development. It would be a platform on which developers could work and deploy further solutions, like robotic arms, racks for luggage or containers for drugs. The analogy of this approach could be an Apple smartphone with AppStore, where developers could create applications and offer them to customers.

1.2. Aims and objectives

This section defines the conditions of successful completion of the project.

1.2.1. Aim

A robotic device designed to function around people requires an outstanding amount of work and expertise to be made and scope of such a development would be too large for a single project. Instead, an alternative approach of designing a minimum viable product that would allow to test the idea and inform future studies was taken. This project aims at building a robotic platform for a personal assistant that would resemble basic human functions of movement, person detection and autonomous following. This will be achieved by creating a two-wheeled self-balancing movement platform and then consecutive development of the computer vision capabilities and decision making algorithms, which collectively should be sufficient in representing the potential of the concept.

1.2.2. Objectives

In order to achieve the vast aim of the project, the following objectives need to be met successfully:

1. To perform the literature review of a similar robotics applications
2. To formulate a mathematical model of the system
3. To design the high level system architecture mimicking a human with similar form factor and sensing
4. To design and build a physical model of the robot, including the choice of the actuators, computational units, sensors and build materials
5. To implement and tune an appropriate controller into the system
6. To implement the camera vision techniques in the robot, with a goal of achieving the spatial awareness
7. To design and implement the behavioural algorithms into the control system
8. To seamlessly integrate the entirety of the control system
9. To validate the performance of the robot in a variety of environments

1.3.Scope

The scope of the project is dividable into two parts. The first one is focused on developing the two-wheeled, self-balancing robot serving as movement platform for a whole development. The dynamics of it would be based on the principle of inverted pendulum with control and actuation executed using PID controller and DC motor drive-train system.

The second part is based on the computer vision and autonomous behaviour. The robot would be equipped with camera serving as an environmental and spatial information input into the system. The camera image would be processed using deep neural networks to detect basic shapes and subsequently a given human. The outputs from computer vision software would be fed into decision making algorithms that would actuate the position of the robot to orientate it towards and keep at a given distance from a followed person.

1.4. Innovation

The components that create the subsystems of the robot are well understood and developed. The novelty of the system lays in the integration of the already existing sub-systems to result in a semi-intelligent robot capable of autonomous navigation. While similar devices were already created, they were either very simplified and low cost or relatively sophisticated and resource exhaustive. This project is differentiated from its predecessors by being at the middle ground of the complexity and cost spectrums.

2. Literature review

This section serves as a theoretical introduction to the fields required to build the robot outlined in the Introduction.

2.1. Two-wheeled, self-balancing robots

2.1.1. Mathematical model and control

As described by Varghese et al, the dynamics of a two-wheeled, self-balancing (TWSB) robot can be accurately approximated by the inverted pendulum system. Inverted pendulum, presented in Figure 2-1, can be thought of as a normal pendulum, which is a lump of as mass suspended from a pivot that can swing freely, that has had its centre of mass displaced above the pivot point. Such position makes the system inherently unstable, as the mass always tends to be below the pivot point at the lowest possible position, so as to arrive at system's energetical minimum. In order to keep the upward position of pendulum the system needs to be dynamically stabilised.

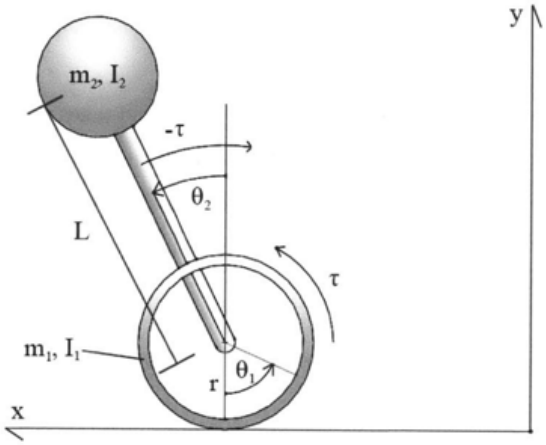

Figure 2-1 Inverted pendulum on wheels model (Bageant 2011)

The stabilisation is achieved by appropriate actuation on the system. The force is exerted indirectly on the pendulum through acceleration of wheels, which connect to the main body via a common axis. The stabilisation occurs as follows: as the pendulum is falling towards the right hand side, the wheels need to be accelerated clockwise, thus creating an inertial force that is counteracting the gravitational pull on the pendulum and vice versa for the opposite direction.

An appropriate controller computing the stabilisation actuation is required for dynamical balancing. Design and implementation of controller is usually based on the equations of motion of the system describing the relationship between the position and its derivatives and externally applied forces. For the inverted pendulum, (2.1) shows the equations of motion derived by Verghese et al in using Lagrangian approach.

$$\ddot{x} = \frac{u+m_p l_p(sin\theta)\dot{\theta}^2 - m_p g cos\theta sin\theta}{M_c+m_p-m_p cos^2\theta}$$

$$\ddot{\theta} = \frac{ucos\theta+(M_c+m_p)gsin\theta+m_p l_p(cos\theta sin\theta)\dot{\theta}^2}{m_p l_p cos^2\theta-(M_c+m_p)l_p} \quad (2.1)$$

It can be noted that the equations are non-linear, which vastly complicates the controller design and limits possible options. A common approach in such situation is linearization of equations around an operating point. According to Bageant, this approach can be taken if:

- The angle of tilt from the vertical of the upper body is very small, so that sin(θ) = θ
- The angular velocity of the tilt of the upper body is sufficiently small, so that the centrifugal force may be neglected
- The linearized model will be used only around the operating point, in this case small angle deviation from the vertical position

The equations can be linearized by a number of methods. 2.2 shows the linearized equations of motion using Jacobian method, derived by Verghese et al. The equations are in state-space form.

$$\frac{d\delta x}{dt} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{(M_c+m_p)g}{M_c l_p} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{m_p g}{M_c} & 0 & 0 & 0 \end{bmatrix} \delta x + \begin{bmatrix} 0 \\ -\frac{1}{M_c l_p} \\ 0 \\ \frac{1}{M_c} \end{bmatrix} \delta u \quad (2.2)$$

Based on the linearized equation a model of the system can be built, which could be used to design and test an appropriate controller. Bageant in 2011 investigated application of various controller types on TWSB systems. Firstly, the proportional-integral-derivative (PID) controller was modelled. It was tuned using root locus shaping method and further improved by application of low lead-lag compensator to decrease low-frequency oscillations and low-pass filter on input data to eliminate the high-frequency oscillations. The resulting controller was capable of dynamically balancing the system, however the overall performance wasn't ideal. In search for the most optimal control an alternative, more complex approach of state space controller design was taken. The advantage of state space controller over PID is that its poles, which are parameters defining the system performance, could be placed or chosen wherever suitable. That allowed to find the most optimal gain matrix defining the controller response using pole placement technique. The state space control gave much better performance compared to PID, with almost non-existing oscillations, however the results were purely

theoretical as the controller was never implemented on a real hardware system. The key outcome of the Bageant study is that there are a few options to choose from regarding controller choice for TWSB robots, with controller's performance increasing along its complexity.

2.1.2. Low-cost projects

The published works regarding two-wheeled self-balancing (TWSB) robots vary in their complexity, cost and sophistication greatly. A relatively big group of published TWSB robots is characterised by their small size, low cost and usage of variation of Arduino microcontroller as a base for computation.

One of the simplest robot designs was proposed by Hau-Shiue and Lum in 2013. The robot is balanced using low cost Arduino Mega microcontroller and small DC motors. The control is performed in closed loop by Proportionate-Integral-Derivative (PID) controller with sensory input from gyroscope and accelerometer. The signal from two sensors is combined using complementary filter, which eliminates both the drift of the gyroscope and the noisiness of the accelerometer and provides stable pitch angle readings. The frame and drivetrain are made from low cost materials such as plastic sheets and metal rods, which collectively create the simplistic construction presented in the left-most of Figure 2-2. Despite its simplified form, the robot is able to balance itself and withstand disturbances, up to the relatively low actuation limit of the DC motors.

Another example is coming from the work published by Hellman et al in 2015 and can be seen at the centre of Figure 2-2. The robot was characterized by more solid structure that allowed for installation of slightly bigger motors. Similarly to Lum et al, the PID controller was implemented, however the data fed into it was filtered using the Kalman filter, which is a much more complicated filter method with more variables to control than the complementary filter and usual give performance superiority. The last example from the low-cost group is the Balanduino, which is a open-source TWSB robot released in 2014 by TKJElectronics firm, presented at the right side of Figure 2-2. Construction-wise it is extremely similar to the other mentioned projects, with a notable distinction in control system that allows for Bluetooth connectivity with a smartphone. The movement control is executed by modification of PID controller in form of varying angle setpoint, which allows for acceleration in a given direction.

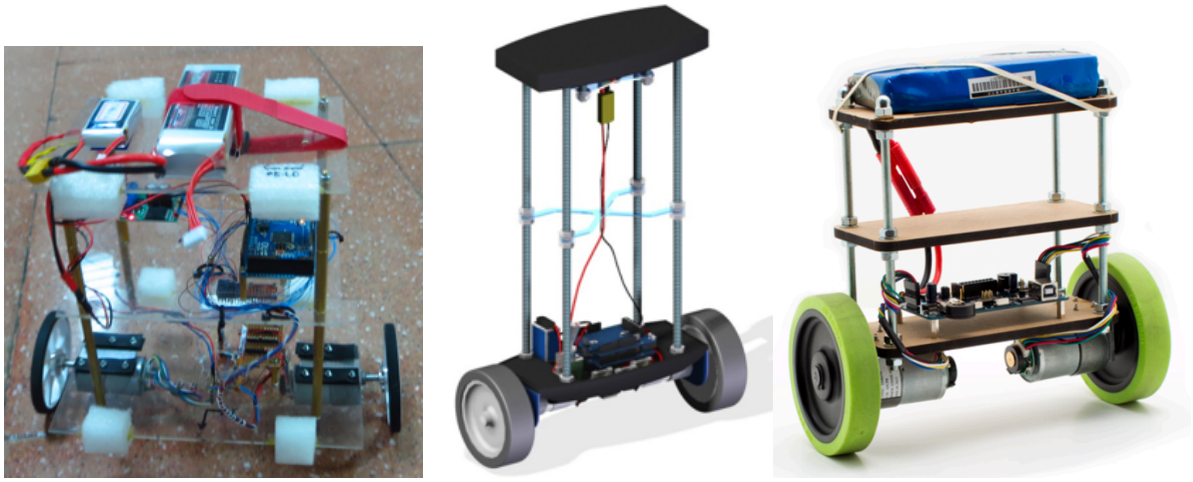

Figure 2-2 Low-cost, Arduino based two-wheeled self-balancing robots. Sources from left to right: Lum et al 2013, Hellman et al 2015 and Balanduino 2014.

2.1.3. High-cost projects

In order to find more sophisticated projects, one needs to search through the commercial sector. There are numerous examples of TWSB robots that achieved commercial success, most notably the human transporter Segway. The Segway transporter is a scaled-up version of TWSB robot that allows the user to step onto the robot and tilt into a desired direction, which results in a controllable movement as the robot is trying to correct the tilt by inducing acceleration. Segway also created Loomo, which is combining the functionality of personal transporter with autonomous person following and voice commanding. Additionally, it can navigate around the house and manipulate the environment through communication with IoT devices and smart home infrastructure. It is visible in Figure 2-3.

Figure 2-3 Segway Loomo assistant robot and personal transporter

Overall, the topics of TWSB robots design and control are well researched. However, there seems to be a gap between very simplified, low-cost solutions and the high-end, commercial grade products. An empty space for commercial-grade, but lower relative cost robot exists. Robot that would be of considerable size and sound structure. Robot that would be equipped with a capable powertrain system, providing actuation required for drive on a range of surfaces and withstanding reasonable disturbances. Robot-wise, this project is aiming at filling that gap.

2.2. Navigation

Giving robots ability to navigate autonomously is a trend initiated decades ago, but thanks to the contemporary advances in computing power the trend has been accelerated and allowed the solutions to be deployed on-the-edge and empower mobile robotics.

There were two notable projects that impacted the design process of the robot outlined in this report and were partially incorporated in the design architecture. The first system was created by Rahman et al in 2016. The physical construction of their robot is similar to those presented in Figure 2-2, with an addition of digital camera and a microphone. The system first captures the voice command through the microphone and extracts the object name from it using Kalman filter and match-making to the database entries. Next, the system searches for the extracted object using the camera input run through the object recognition algorithm. Once the object is recognised, its centroid location is captured and the robot is set to follow the object at a given distance as long as it can be found in the front view. The object recognition steps are presented in
Figure 2-4.

Figure 2-4 Object recognition performed by the robot designed by Rahman et al in 2016. Consecutive steps: capture, shift to black and white and object detection through edge detection

The other project was published by Ghani et al in 2011. They have focused on line-following capabilities achieved by utilisation of infra-red (IR) sensors. The logic implies that the line must always be underneath the robot within a given placement. The position of the robot relative to the line is detected using a set of IR sensors. The systems logic and detection components are visible in Figure 2-5.

Status	Sensor Input			Output
	L	M	R	
2 cm	0	1	0	FORWARD
	1	1	0	SHIFT TO LEFT
	1	0	0	FAST SHIFT TO LEFT
	0	1	1	SHIFT TO RIGHT
	0	0	1	FAST SHIFT TO RIGHT

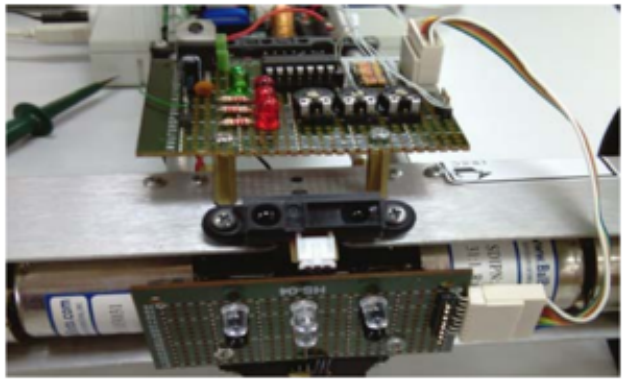

Figure 2-5 Left: navigation logic. Right: a set of infra-red line-detecting sensors. (Ghani et al 2011)

2.3.Computer vision

As shown by the previous section, there could be different approaches to providing robots with navigation information. Besides the presented approaches, one could also list radar-based, global positioning system (GPS)-based or ultrasonic-based navigation. However, no single navigation system has as much flexibility, low cost entry barrier and potential connected with modern computing powers as utilisation of a camera feed in form of computer vision.

Computer vision (CV) is an interdisciplinary scientific field that deals with how computers gain high-level understanding from digital images or videos in purpose of understanding and automating tasks that the human visual system can do (Wikipedia 2020). Computer vision task include methods for acquiring, processing, analysing and understanding digital images and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, for example in forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that make sense to thought processes and can elicit appropriate action.

Sub-part of the CV that deals with finding and identifying objects and people in an image or video frame is called object recognition. There are numerous approaches to object recognition, with distinctions being made in the architecture, power requirements, speed and accuracy. One of the most popular modern methods is based on the artificial neural networks (ANN) and its details are explained in the Section 2.3.1

2.3.1. Deep Neural Networks

Most of the methods applied to object recognition fall into either machine learning based approaches or deep learning based approaches. In the category of machine learning, it is necessary to first extract the features of an object and then separately perform the classification. The deep learning approaches are able to perform end-to-end object recognition without specifically defining the object's features. In general, the deep learning approaches are more suited to detection a wide range of objects, while machine learning approaches are more focused on a particular group or feature. Additionally, the deep learning approach requires much more time to train due to sheer amount of variables that need to be set, however its accuracy and speed scale much more beneficially with growing amount of data (Sharam 2017). As there are numerous datasets available online, such as COCO which includes over 300 000 images of various objects, including humans, the deep learning approach gives more promise.

It was decided to utilise the deep learning approach in this project due to its flexibility in the acceptable inputs and capability to detect other objects than human, which may be useful in later stages of development. Deep learning is based on the deep neural network, which is a subtype of artificial neural network.

Artificial neural networks (ANN) are computing systems which architecture was inspired by biological neural networks that constitute animal brains. Such systems are able to learn, or progressively improve their ability, the same way that human does. In case of object detection, a set of labelled images, for example, "dog" or "no dog" is fed into the system, which then analysis the pictures and progressively learn to recognize those that contain dogs. This is done without explicitly programming such system with particular task, but rather letting the algorithm to create its own workings.

ANN consist of units called artificial neuron that are connected together using synapses. Each neuron is receiving the data from upstream and it's processing it according to its state. Both neuron and synapses can have weights, by which the received data is multiplied. The weights of neurons and synapses are the parameters that adjusted during algorithm learning – similarly to the processes that occur in our brains.

Neuron are usually organized in layers. Although the exact action of each layer isn't specified, different layers usually perform different kinds of transformations of their inputs. Deep neural networks (DNN) are a subtype of ANNs. The architecture of DNNs was described by Bengio in 2009 and it is usually characterized by multiple layers between the input and output. Due to additional layers when

compared to ANNs, the DNNs are able to model complex non-linear relationships. Extra layers allow for composition of features from lower layers and thus exponential increase in the complexity that can be extracted from an input. Example of a DNN is shown in Figure 2-6

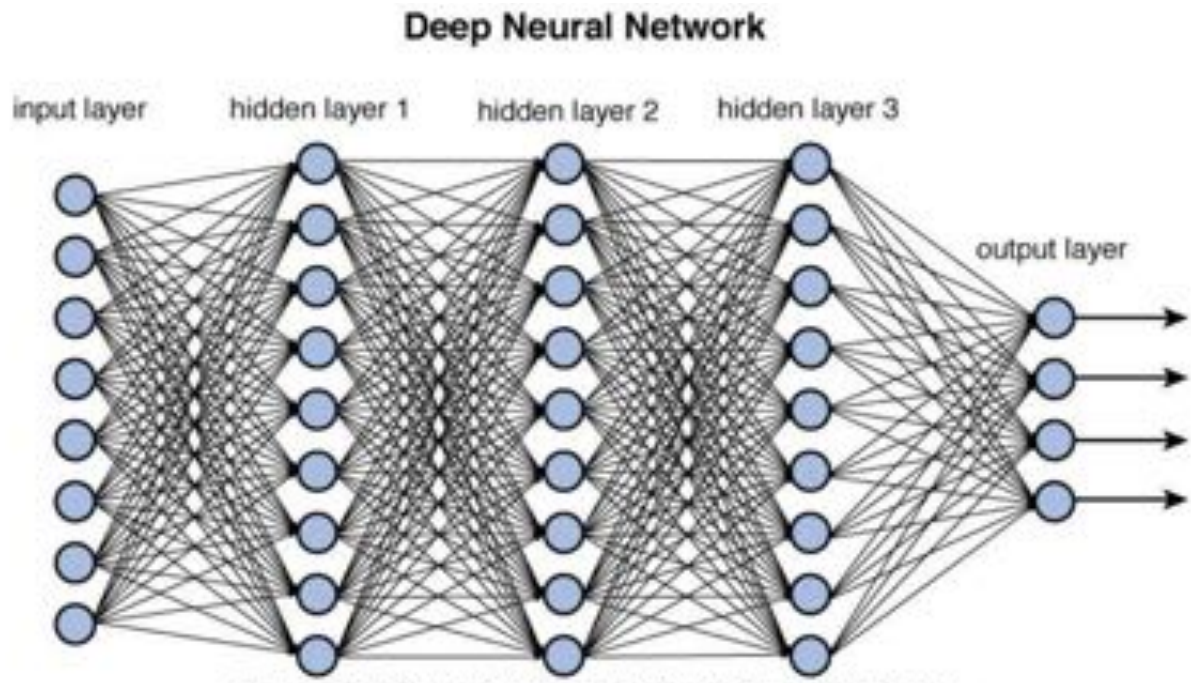

Figure 2-6 Deep neural network schematics (Kyrykovych 2020)

2.3.2. Object detector

Inside of the deep neural network subclass of object detectors, there are numerous solutions with varying architecture and applications. As of 2020, the most popular architectures:

- Region Proposals (R-CNN, Fast R-CNN, cascade R-CNN
- Single Shot MultiBox Detector (SSD)
- You Only Look Once (YOLO)
- Single-Shot Refinement Neural Network for Object Detection (RefineDet)
- Retina-Net
- Deformable convolutional networks

All those architectures differ with strengths and weaknesses. For a mobile computing application, which this robot is, the object detector must be first and foremost fast and computationally inexpensive, so that real-time detection on mobile computing unit can be achieved. Accuracy of detector is secondary, as all of the architectures achieve reasonable levels. Accuracy and speed are usually compromised against each other, therefore a reasonable balance must be find. Finally, the

architecture must require its output to be within the range of the onboard mobile camera. In order to choose the appropriate solution, the results of LPIRC were reviewed.

The Low-Power Image Recognition Challenge (LPIRC) is an annual competition started in 2015. The competition identifies the best technologies that can classify and detect objects in images efficiently and accurately, with a special highlight on short execution time, low energy consumption and high precision. The 2018 edition featured three categories, each with increasing computing powers. Hardware from category 2 was the most similar to what the robot is ought to possess, therefore outcomes only from that category were taken into consideration. It was found that two architectures, namely YOLO and MobileNet SSD yielded the best results, in terms of trade-off between speed, accuracy and energy consumption. Both architectures are based on the same working protocol, namely single shot detector (SSD), which inputs the whole image at once at performs both detection and classification in parallel. That method results in faster and less computationally intensive object recognition as compared to scanning the picture and performing operations line by line or sector-based. As MobileNet SSD is much easier network to be deployed and worked with, it was chosen to be proceeded with.

3. Design methods

This section describes the methodology of executing the aim of the project through achieving the objectives while using the findings from literature review

3.1. Top-level design

This section details the design process utilised to generate the top-level architecture of the robot

3.1.1. Inspiration

Nowadays we have numerous software personal assistants within our reach, like Siri or Alexa just to name a few. However, those entities are purely virtual and only capable of manipulation of information. The inspiration behind this project was to create a tangible personal assistant that could manipulate both information and matter. It could be equipped with software capabilities like Natural Language Processing or web search via utilisation of one of the mentioned assistants, but also capable of physically existing within human space and being able to manipulate it and have real influence over it.

Figure 3-1 R2-D2 droid, one of the characters from Start Wars franchise (Wikipedia 2020)

The concept was already portrayed decades ago, with one of the most prominent fictional examples being droid R2-D2 from *Star Wars* (Figure 3-1). Obviously fictional, it is still a great example of how robotic assistants could be utilised in human environment. The idea of the robot designed and built in this project is largely inspired by R2-D2, with similar dimensions, movements capability and environmental awareness, thus trying to bring the fictional and real world ever so slightly closer.

3.1.2. In the image and likeness of human

Robotic assistant is an artificial system designed to function effectively within human environment, therefore it would be wise to learn from the very beings natively operating in that environment. Human can be thought of as a very complicated and sophisticated machine that went through millennia of research and development activity by the process of evolution. It would be foolish to rejects learning done by the nature when designing a machine that is to operate within human conditions and environment.

3.1.2.1. Form factor

One of the first things to consider in terms of the structure is the form factor. Humans have specific, elongated form factor, with its upright position and relatively closely spaced legs as showed few millennia ago by Leonardo Da Vinci in his Vitruvian Man drawing, presented in Figure 3-2. The robot shall have similar form factor in order to fit in corridors, doors and tunnels, but also be able to reach to surfaces or appliances.

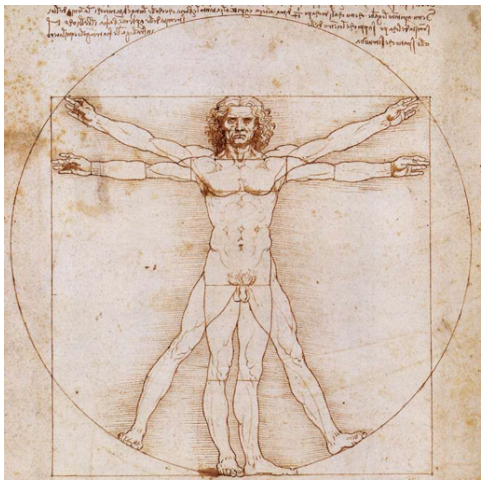

Figure 3-2 Leonardo's Da Vinci "Vitruvian Man" drawing showing proportions of human body (Wikipedia 2020)

3.1.2.2. Mechanics of human walking

Human walking can be accurately represented as movement of inverted pendulum. It is a system which has its centre of mass located above its pivot point. The mechanics of every instance of a human taking a step is like inverted pendulum – we take our self out of the balance position of standing, let our body move or fall towards the direction in which we are travelling and then we dynamically balance our self by taking the step and retaining our centre of mass directly above the pivot point, which is usually located at our feet. With the form factor proposed in the previous section, the robot shall mimic the movement mechanics of the inverted pendulum for optimal movement capabilities. Figure 3-3 shows the human walking mechanics as inverted pendulum

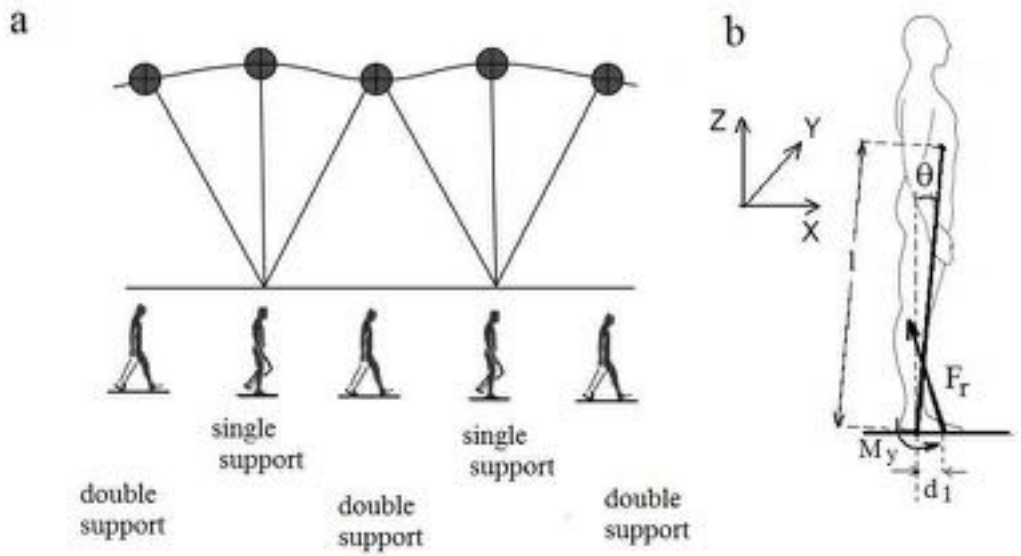

Figure 3-3 Human walking mechanics represented as inverted pendulum (Zielinska et al 2017)

3.1.2.3. Visual perception model

Visual perception is probably one of the human most sophisticated sensory systems. It allows us to capture the information about surroundings in form of electromagnetic waves and extract our position in relation to the surrounding, but also object recognition and behavioural information. According to Peters and Pal, the visual perception model is constructed as follows:

" …'seeing' consists of mappings from sense inputs from sensory units in the retina of the eye to cortex cells of the brain stimulated by sense inputs. A sense input can be represented by a number representing the intensity of the light from the visual field (i.e., everything in the physical world that causes light to fall on the retina.) impacting on the retina. The intensity of light from the visual field will determine the level of stimulation of a cortex cell from retina sensory input. Over time, varying cortex cell stimulation has the appearance of an electrical signal. The magnitude of cortex cell stimulation is a real-value. The combination of an activated sensory cell in the retina and resulting retina-originated impulses sent to cortex cells (visual stimulation) is likened to what Poincar ´e calls a sensation in his essay on separate sets of similar sensations leading to a perception of a physical continuum (Poincar´e, 1913 & 2009) "

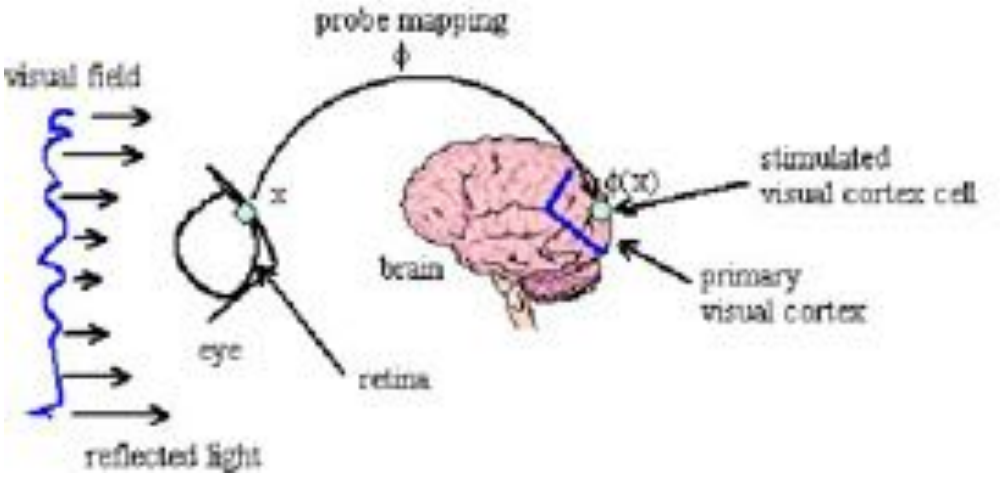

Figure 3-4 Model of the human visual perception (Peters and Pal 2010)

Therefore, all the required information required for spatial awareness and autonomous person-following can be extracted from the environment using vision techniques. Thus, the architecture of the robot vision capabilities should resemble that of a human, which is possible by application of the Neural networks and camera system described in Section 2.3

3.1.3. Design criterion

In order to ensure these initial drafts would meet the aim and objectives set out by this report a design criterion was set out, considering the findings of the literature review:

Considering the aims and objective set by this report and the findings from the literature review, the following design criterion was set out:

- The design must take form factor of human being
- The mechanics should be similar to that of human being, meaning being based on the inverted pendulum principle with dynamical balance capability
- The design must be equipped in the powertrain capable of exerting forces required for the mechanics model
- The comprehension of the environment, which allows for spatial awareness and person recognition and tracking should be based on a visual perception model
- The structural build should be robust, allowing for bearing loads exerted by both the external disturbances existing in the human environment and the internal powertrain-induced forces
- The robot should be capable of driving through a range of terrain and over reasonably-sized obstacles, that can be encountered in the day-to-day human activitiy

3.1.4. Generated concept

Based on the above sections, an initial concept sketch of the two-wheeled robot was produced, visible in Figure 3-5. It was used as the reference for CAD model presented in the next section

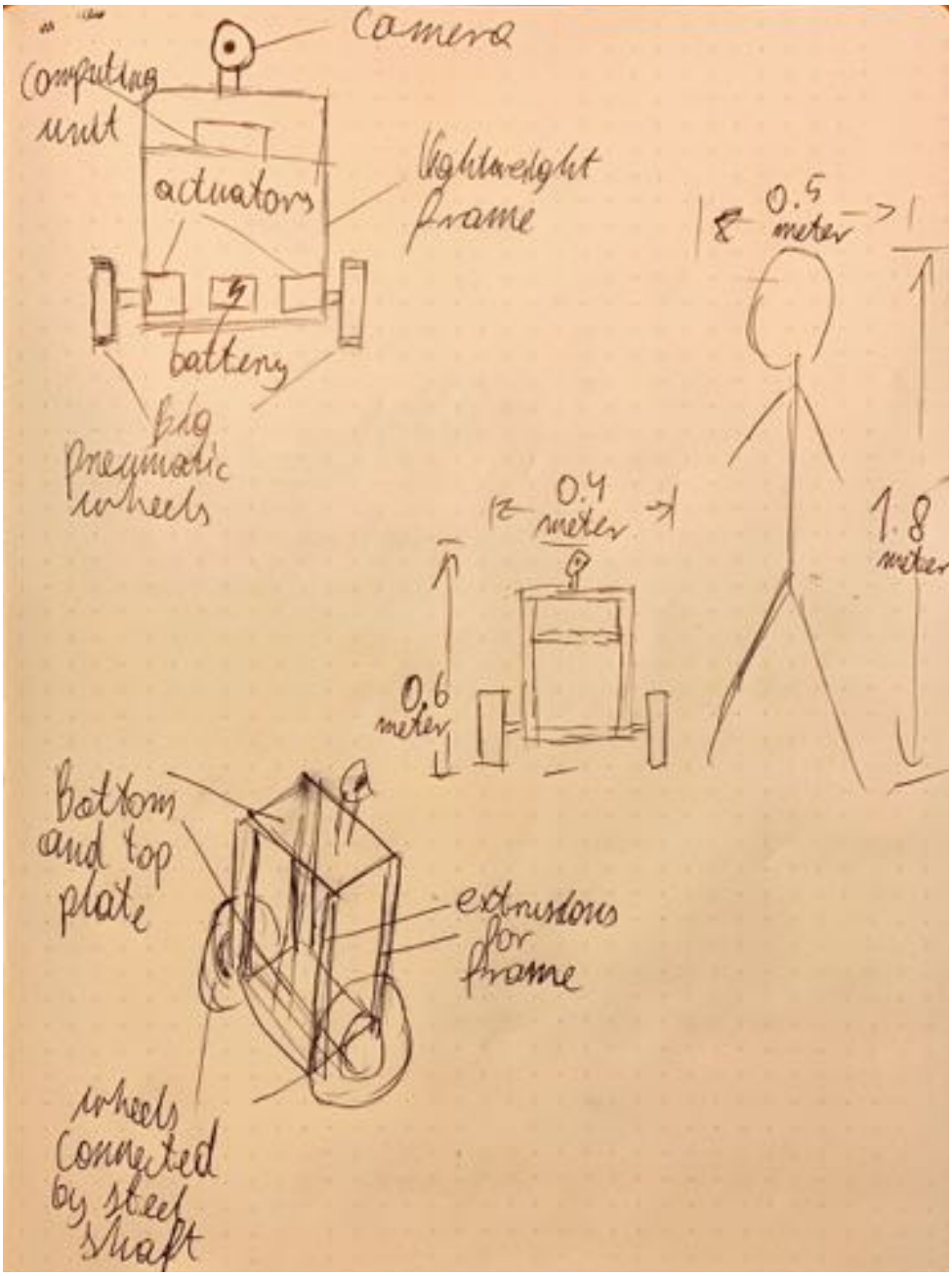

Figure 3-5 Sketch of the design concept

3.2. Electro-mechanical design

This section details the mechanical and power design of the robot. Key elements are discussed and particular choices justified

3.2.1. Actuators choice

Actuators are crucial components in this robotic systems and their choice will lead the next design steps. In order to dynamically balance the inverted pendulum system, an appropriate force need to be exerted on it that would accelerate the system in a given direction. The force will be exerted on the system through the wheels that have contact with ground. However, the particular method of driving the wheels has a great impact on the overall performance of the system.

The actuators required for the two-wheeled, self-balancing robot have to meet the following requirements:

- The actuator's output need to be a rotary motion
- The actuator needs to be connected to a wheel
- The actuator needs to have high torque at low revs
- The actuator needs to be able to sustain high angular velocities for an elongated amount of time
- The actuator need not to create extensive amounts of heat for durability reasons

There are primarily two kinds of actuators that could be utilised in this system, namely Direct Current (DC) motor and stepper motor.

Table 3-1 Comparison between Stepper Motor and DC Motor

	Stepper Motor	**DC Motor**
Advantages	-High torque, constant along revolutions spectrum -Lower price	- Torque varying across the revolutions spectrum, with minimal values located at low revs -Can operate at high angular velocities for extended amount of time -Do not produce extensive amounts of heat
Disadvantages	-Limited max angular velocity -Cannot operate at high angular velocity for long time -Considerable heat up during operation	- Low torque at low revolutions per minute value -Higher price

Based on the comparison presented in Table 3-1, it was decided to use DC motors due to their higher tolerance to high angular velocities and possible longer life span. However, due to the characteristic of producing low torque at lower revolutions per minute, an additional gearing system was required in order to convert the high angular velocity into high torque.

3.2.2. CAD design

Based on the rough sketches presented in Section 3.1.4 the detailed CAD model was created using SOLIDWORKS software. The outside dimensions of the robot were decided to be 600x300x200mm. The robot's CAD design is showed in Figure 3-6, Figure 3-7 and Figure 3-8. The annotations from those figures are explained in Table 3-2

Table 3-2 Bill of Materials of the robot

ID	Name	Quantity	Manufacturing process	Material
1	Rexroth Extrusion	4	Cut to length using saw	Aluminium
2	Top front/back plate	2	Laser cut	Aluminium
3	Bottom front/back plate	2	Laser cut	Aluminium
4	Electronics plate	1	Laser cut	Aluminium
5	Upper side plate	2	Laser cut	Aluminium
6	Lower side plate	2	Laser cut	Aluminium
7	Motor plate	2	Laser Cut	Aluminium
8	Motor plate rail	4	3D printing	PLA
9	Pneumatic wheel	2	Purchased	Rubber
10	16mm shaft	1	Cut to length using saw	Steel
11	Big timing pulley	2	Laser cut + glue	MDF
12	Shaft holder	2	3D printing	PLA
13	Belt tension assembly	2	3D printing	PLA + steel
14	DC motor -	2	Purchased	Misc
15	Rotary encoder	2	Purchased	Misc
16	Encoder plate	2	3D printing	PLA
17	Enc. plate spacer	2	3D printing	PLA
18	Small timin pulley	2	Purchased	Aluminium
19	Secondary DC motor shaft	2	3D printing	PLA
20	DC battery	1	Purchased	LiPo
21	Electronic speed controller	1	Purchased	Misc
22	CPU - Raspberry Pi	1	Purchased	Misc
23	Microcontroller Teensy	1	Purchased	Misc
24	Intertial measurement unit - MPU 6050	1	Purchased	Misc
25	Pi Camera	1	Purchased	Misc
26	Pan-Tilt Hat	1	Purchased	Misc
27	Camera plate	1	3D printing	PLA
28	Bottom electronics holder	1	3D printing	PLA

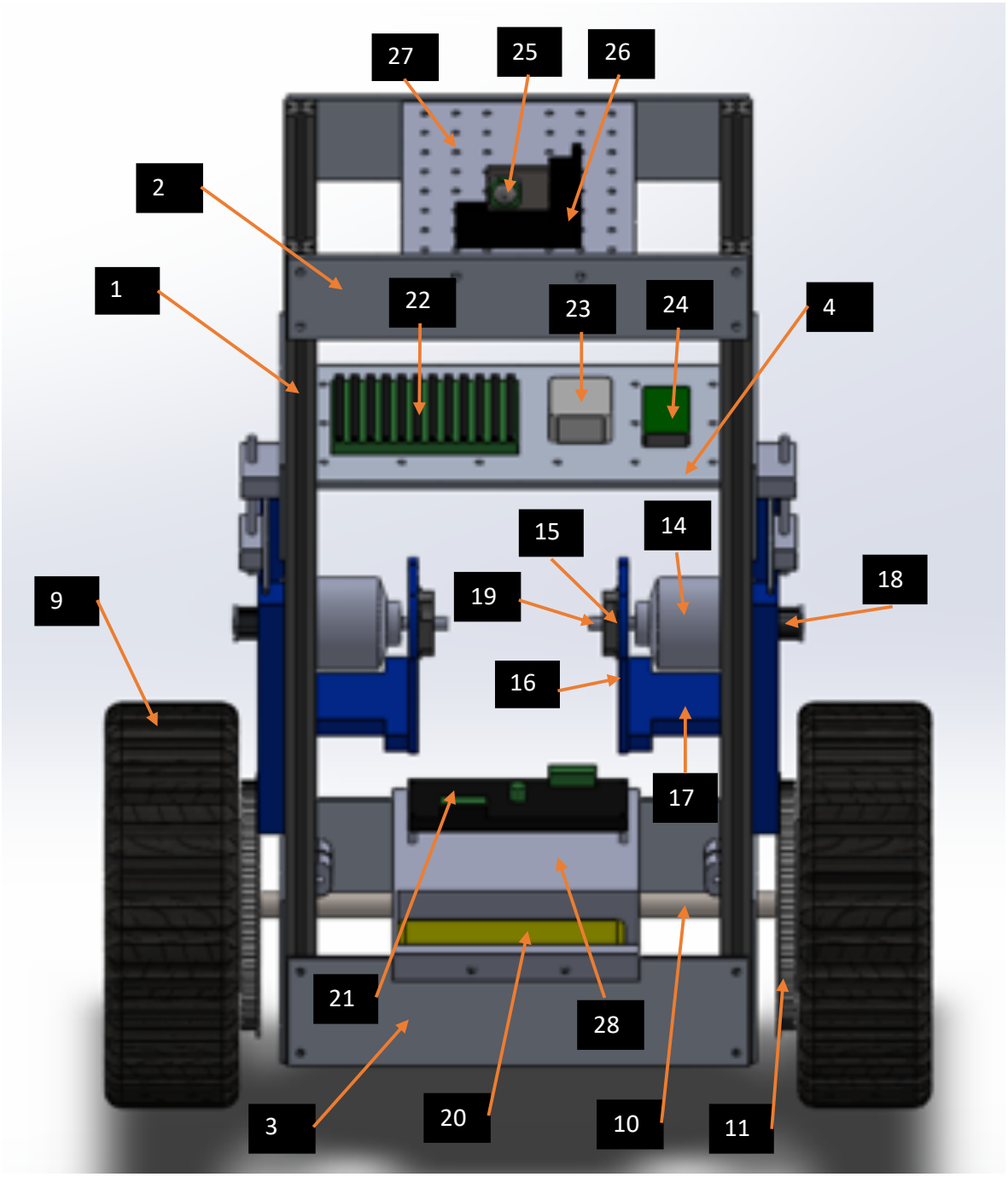

Figure 3-6 Robot CAD design: front view with annotations

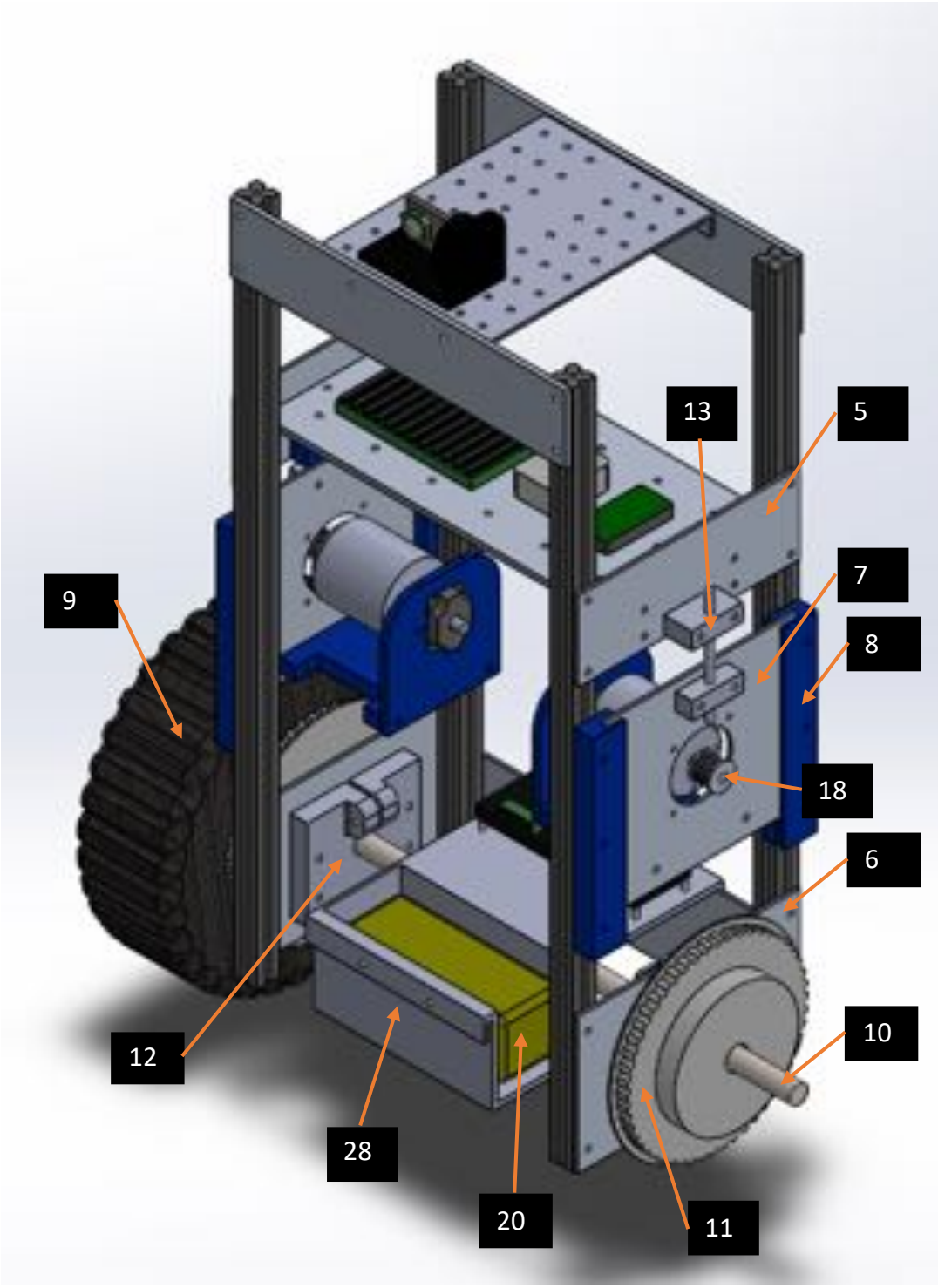

Figure 3-7 Robot CAD design: Side view with annotations

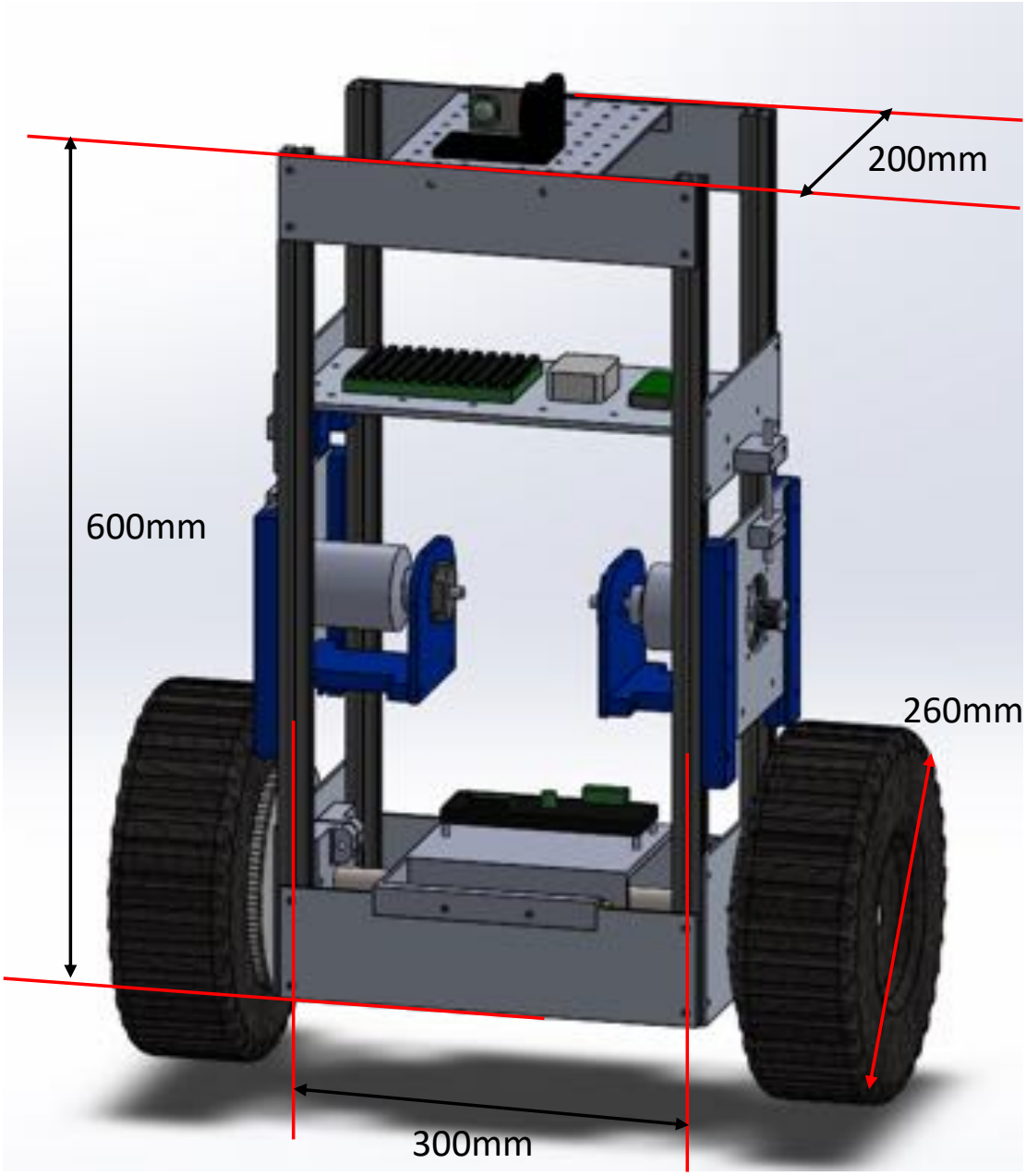

Figure 3-8 Robot CAD design: Angled front view with outer dimensions

The CAD design consists of numerous subsystems and structural components. The role of all parts and subsystems is described in the sections below, outlining the reasoning between crucial design decisions

3.2.3. Frame

Frame is one of the most important parts of the robot, as it serves as main structural component effectively bearing majority of forces. It is important for the frame to be lightweight, stable and durable. It was decided for the robot frame to be made out of two types of elements: aluminium extrusions and aluminium plates. The extrusions provide a stable upright construction, whereas the plates serve as connectors between the extrusions and stabilise all torsional and bending DOFs.

Figure 3-9 Two Bosch Rexroth aluminium extrusions connected using V-sliders and 90 degree profile (PacificIntegrated 2015)

The extrusions used are Bosch Rexroth 4mm thanks to their easy cross-connection ability with V-sliders. The plates were made out of 3mm aluminium sheets cut to shape using laser cutter.

3.2.4. Drivetrain

As outlined in Section 3.2.1, the actuator were chosen to be DC motors, which requires addition of reducing gearing system in order to achieve high torque at low angular velocities. There are a few ways to do it, namely:

- Chain and spur
- Pulley and belt
- Timing pulley and timing belt

Due to low coast and high availability of the parts, it was decided to follow on with timing pulley and timing belt gear assembly, similar to the example shown in Figure 3-10.

Figure 3-10 Example of timing pulley and timing belt assembly

The smaller pulley was mounted onto the DC motor shaft and the bigger pulley was bolted directly to the wheels. The wheels were fit onto the 16mm steel shaft using bearing for freedom of rotation. Then, the steel shaft was connected to the main frame using 3D printed holder parts that would keep the shaft immobile.

The timing pulley gear assembly had the following properties:

- Small pulley: 15 teeth, made out of Aluminium
- Big pulley: 200 teeth, cut in slices from 4 MDF using laser cutter, later stacked up and glued together
- Reduction ratio: ~14:1
- Timing belt: GT2 10mm width
- Rubbered pneumatic wheels, diameter: 260mm

Figure 3-11 Robot's drivetrain timing pulleys + timing belt assembly

3.2.5. Powertrain

Following the decision made in Section 3.2.1, the motors were chosen to be Turningy 6364 213kV 3-phase DC motors, one for each wheel. Those motors can be powered by high currents of up to 65 Amps, have the desired velocity-torque characteristics and a relatively low kV rating, which combined with gearing should be enough to provide high torque at low speeds.

The motors cannot be driven directly by the microcontroller due to high current demand. Thus, an additional components needed to be utilised for driving the motors – an electronic speed controller (ESC). The ESC was chosen to be the ODrive v3.6, which allows for very high precision motor drive and execution of even the smallest movements, which helps to match the output resolution of the control system. Additionally, ODrive allows to drive the DC motors in fashion similar to stepper motors, thus enabling extra accuracy at low revs

In order to draw high currents required to perform strong actuation, a battery with high capacity, low-resistance and capability to output high bursts of current was required. Thus, the Turningy Heavy Duty 5000mAh 6S 60C Lipo Pack was used. It is rated at 22V and can supply the discharge of up to 120C.

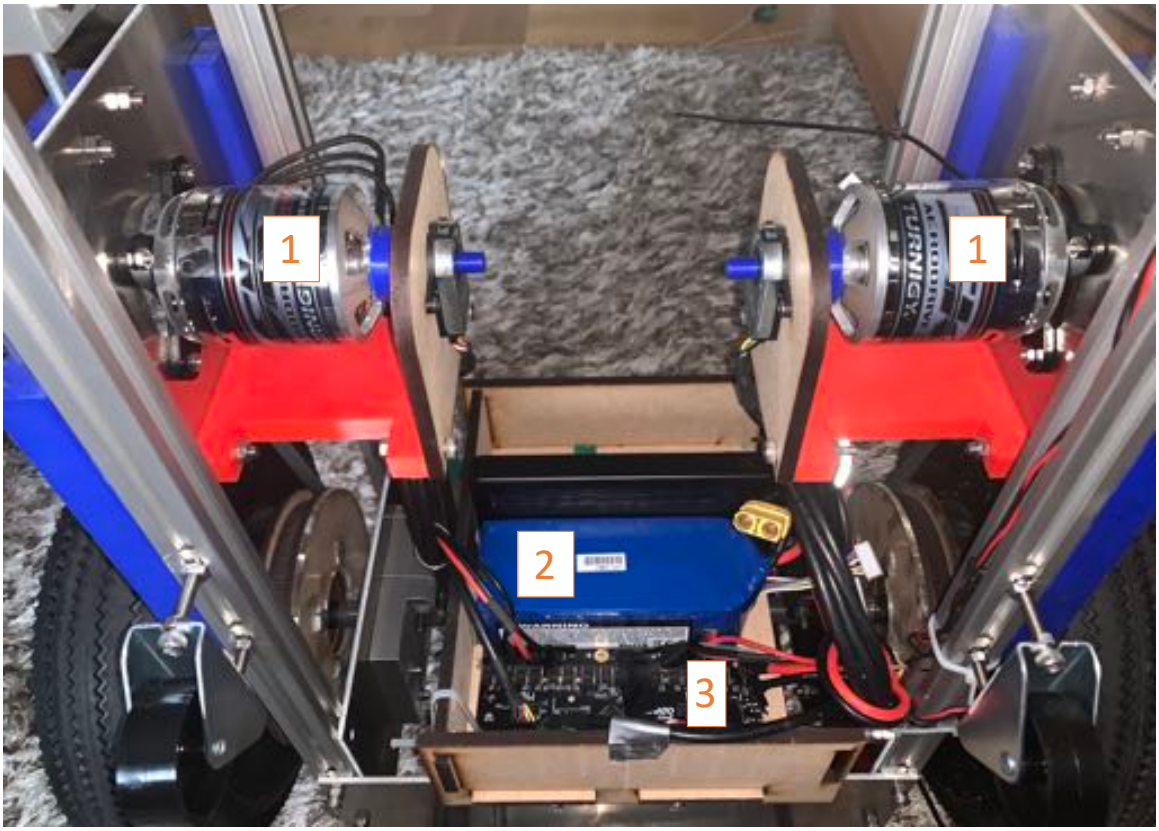

Figure 3-12 Robot's powertrain. 1: Turnigy 6364 213kV DC moto, 2: Turnigy Heavy Duty 5000mAh 6S 60C Lipo Pack, 3: ODRIVE v3.6 high precision motor controller.

3.3. Physical build

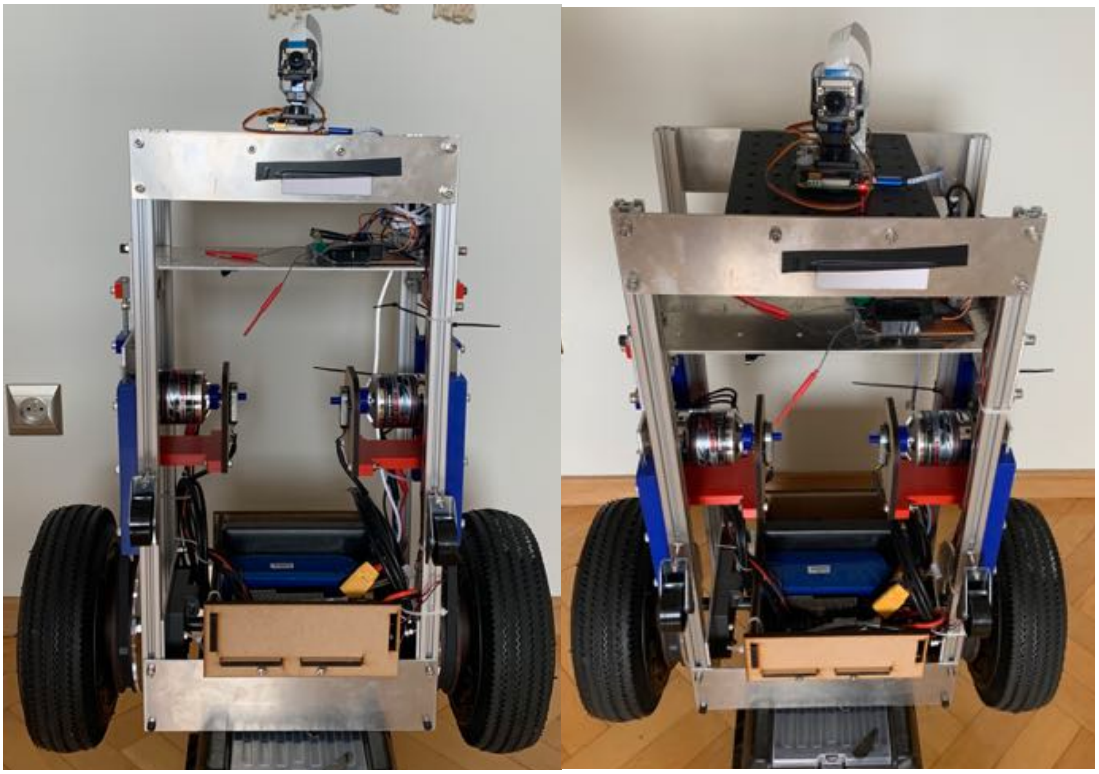

Figure 3-13 Front view of the robot

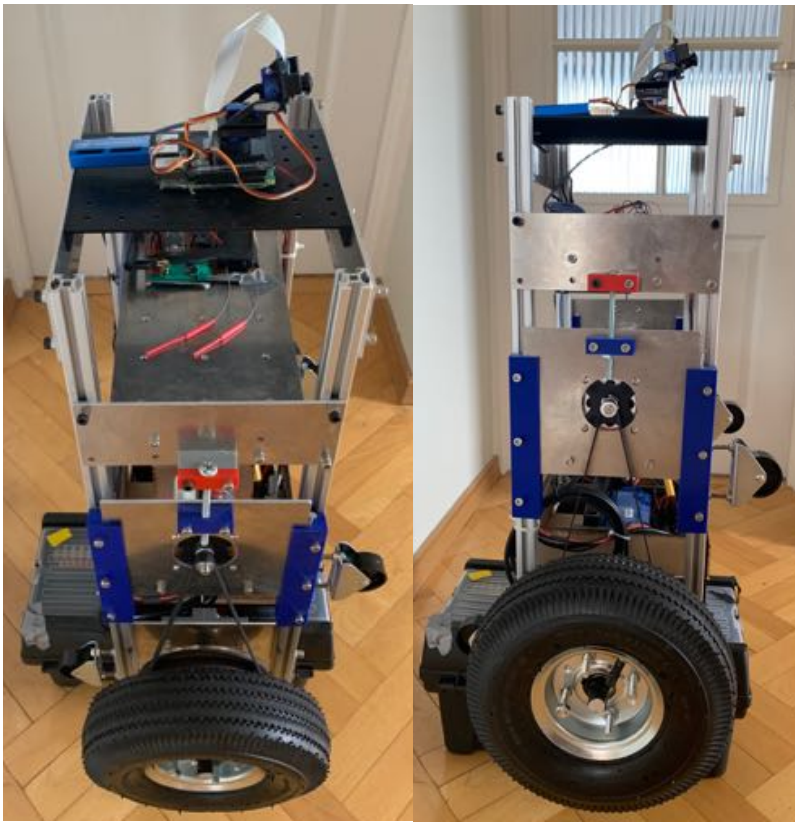

Figure 3-14 Side view of the robot

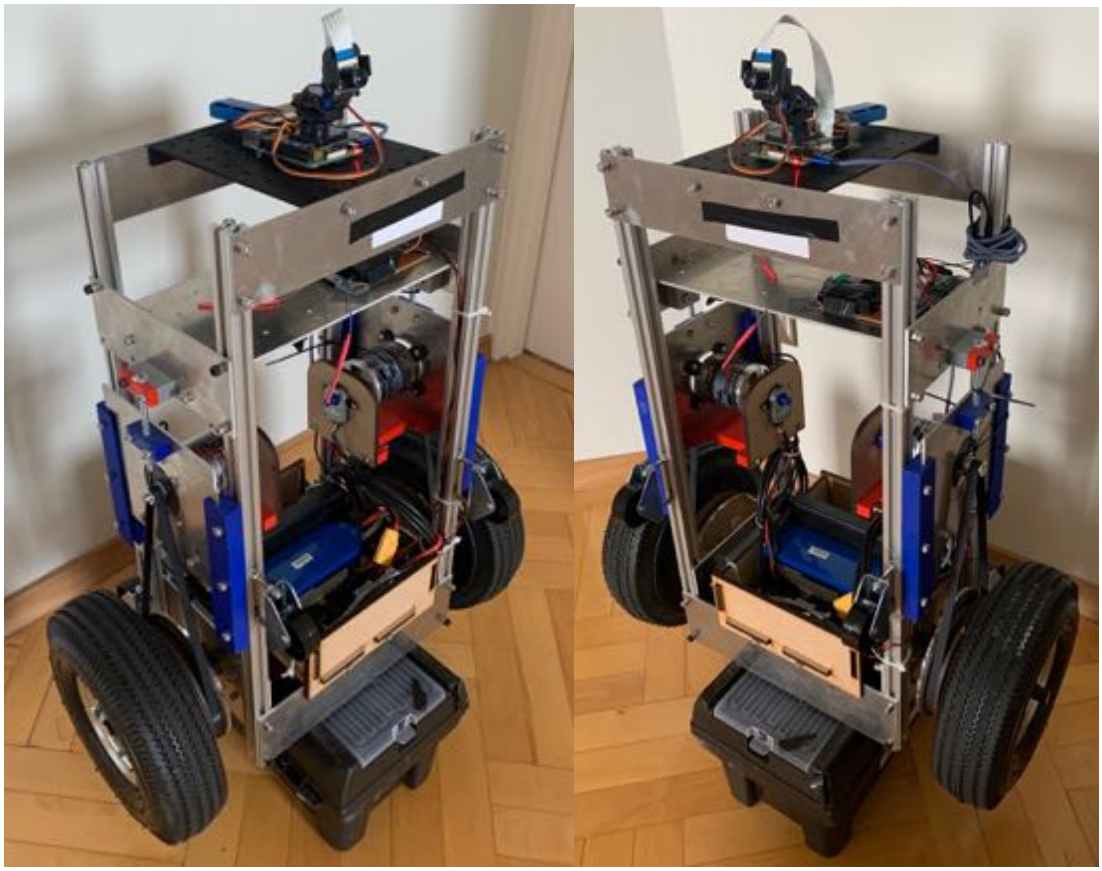

Figure 3-15 Diagonal view of the robot

Table 3-3 Physical properties of the robot

Weight	Height	Width	Depth	Battery cap.	
11 kg	600mm	300mm	200mm	5000 mAh	

3.4. Electronic design

3.4.1. Microcontroller

Microcontroller can be called a “brain” of the robot. It needs to be quick enough to process the data coming from the sensors, manipulate it accordingly using the controller code and output the actuation signals that are to be fed into the ESC that in turns actuates the wheels.

Initially, the first prototype of the robot was equipped with Arduino Uno microcontroller, which can be seen in Figure 3-16. It is one of the most popular devices of its kind on the DIY market. Its operation is very simplistic with numerous digital and analog I/O pins for peripheral connectivity and the included Integrated Development Environment (IDE), which utilises a sub-version of C programming language for coding and ease of compilation. However, it is greatly limited in terms of its computing power by usage of the ATmega328P microchip which is rated at only 16MHz. The initial tests showed unacceptably large refresh rate of the actuation control that was leading to the system instability.

The search for more computing power resulted in choosing the Teensy 3.6, presented in Figure 3-16. This microcontroller board is utilising the same IDE as Arduino and has similar I/O pins selection, however it features a 32 bit 180 MHz ARM Cortex-M4 processor. That hardware swap allowed to decrease the refresh rate to around 10 milliseconds, which greatly improved the performance of the robot and allowed for stable self-balancing. This shows the importance of computational delays and providing short enough sampling rate in discrete systems

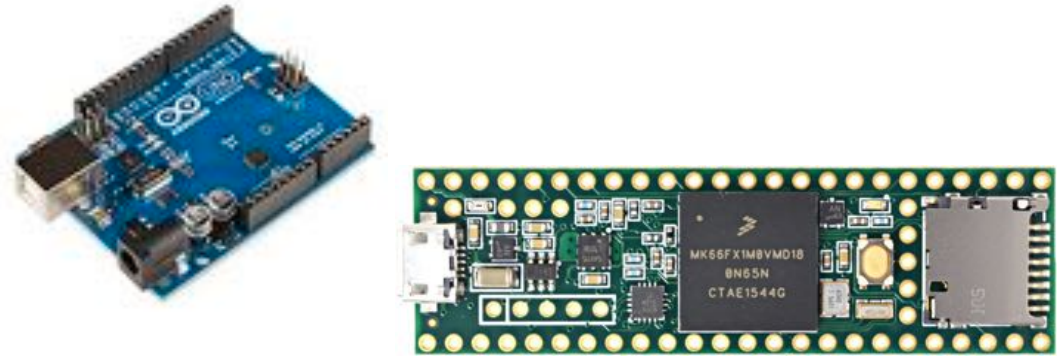

Figure 3-16 Microcontrollers used in the design process. Arduino Uno (left) and Teensy 3.6 (right)

3.4.2. Sensors

The dynamic balance of the robot required a set of sensors as input to the control system

3.4.2.1. Inertial measurement unit – MPU6050

Any self-balancing robot requires data about its inclination in relation to a given axis. This type of data is usually provided by a Inertial Measurement Unit (IMU), which is a device that measures body's specific force, angular rate and orientation using a combination of accelerometers and gyroscopes (Pao 2018). For this application, the MPU 6050 sensor was used, due to its low cost and a reasonable resolution, which features both the accelerometer and gyroscope. The sensor is presented in Figure 3-17

The usual problem that occurs when the accelerometer and gyroscope are used is that the first one is very sensitive to subtle changes and its outputs can be very noisy, and the second one is drifting over time with the measurement value, especially if the body is stationary. However, it happens that the reading of both sensors could be combined together at a pre-defined rate in order to remove the disadvantages of each of them. That approach is called a Complementary Filter. According to the datasheet of the MPU 6050, it is utilising a built-in complementary filter with mixing rate of 97:3 for gyroscope and accelerometer respectively.

Figure 3-17 Inertial Measurement Unit MPU 6050

Overall, the reading obtained using MPU 6050 are very high quality. The sensor reading is stable, as presented in Figure 3-18. In that case the robot was put on the flat platform and the reading of the sensor oscillated only about 0.04 degree. At the same time, the sensor is able to provide high resolution and accuracy of readings, as presented in Figure 3-19, where the robot was oscillated energetically by hand to the pitch of up to 20 degrees. Figure 3-20 represents the data gathered when the robot was tilted up to 80 degrees. Even though the range in that trial was quite extreme, the sensor managed to capture all the dynamics without any drift.

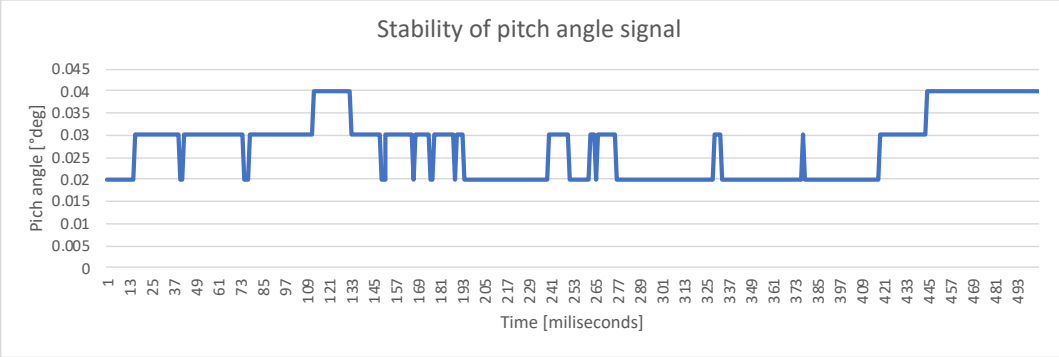

Figure 3-18 Stability of the MPU 6050 readings

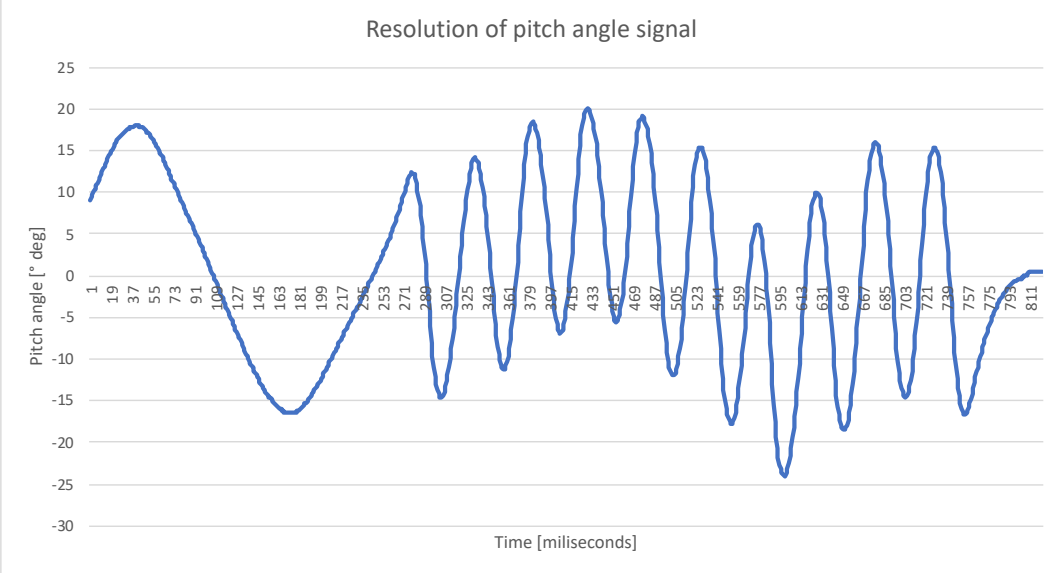

Figure 3-19 Resolution of the MPU 6050 sensor readings during energetic oscillation

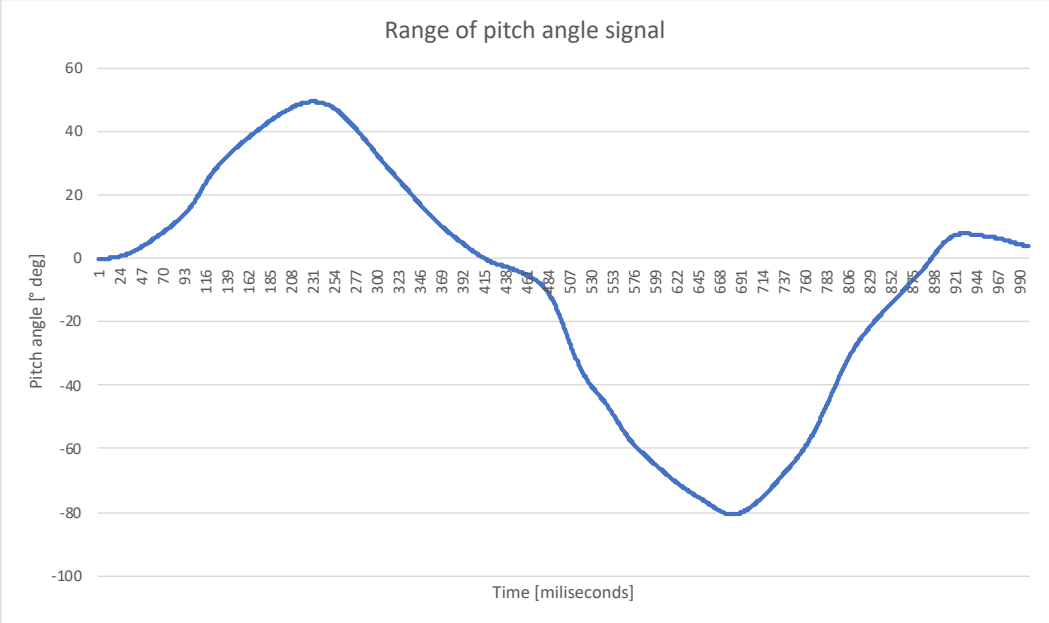

Figure 3-20 Wide range of the MPU 6050 readings

3.4.2.2. Rotary encoder – CUI AMT102

The other sensor that was used in the robot is the rotary encoder mounted to the DC motor. It is providing information about the position of the wheels over the time. Gathering that data allows to calculate the momentary velocity of the robot, determine its past and current position and also predict the immediate future behaviour. It is therefore an useful tool for robot's position tracking, however its main role is to provide information about the motors position for the ESC in order to increase the drive precision. The CUI AMT102 capacitive encoders were chosen due to their high 8192 cycles per revolution (CPR) resolution and the capability to read the angular velocity of up to 7500 rotations per minute (RPM).

Figure 3-21 CUI AMT102 rotatory encoder (ODrive 2020)

Figure 3-22 represents readings from the rotary encoder obtained during self-balancing phase. It can be seen that there is some noise on the signal due to signal oscillation, which may not be representative of the real world situation. Overall, the readings from the rotary encoder are of high resolution and quick response time. The slight oscillation of the values can be reduced using a low-pass filter, which should also improve the stability of the robot

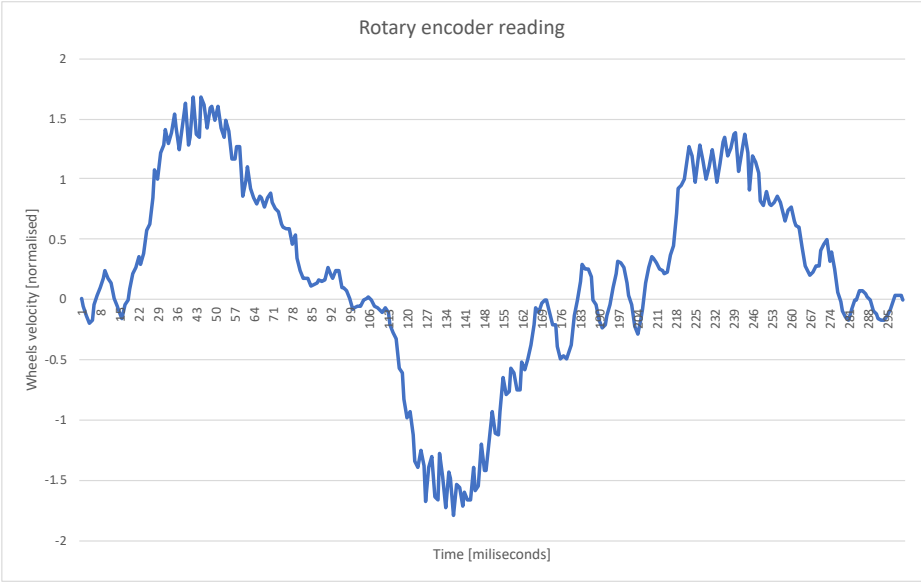

Figure 3-22 Rotary encoder readings from self- balancing phase

3.4.3. Communication

For the purpose of safe operation and increased functionality, it was decided that the robot needs to be radio controlled (RC). Initially, the goal was to build the complete RC system, which was only partially successful due to unforeseen communication protocol issues.

The designed system was based on the 2.4 GHz range signal emitted and transmitted by a pair of nRF24 transceivers, which were connected to Arduino Nano microcontrollers. The controller was designed with two position knobs and 5 switches that would be responsible for on/off functionality of the particular robot subsystems. The controller and the nRF24 modules can be seen in Figure 3-23. However, due to poor quality of the nRF24 connectivity and relatively big computational delays caused by its working protocol, the controller wasn't functional with the high refresh rate of the Teensy microcontroller. Thus, different approach was taken.

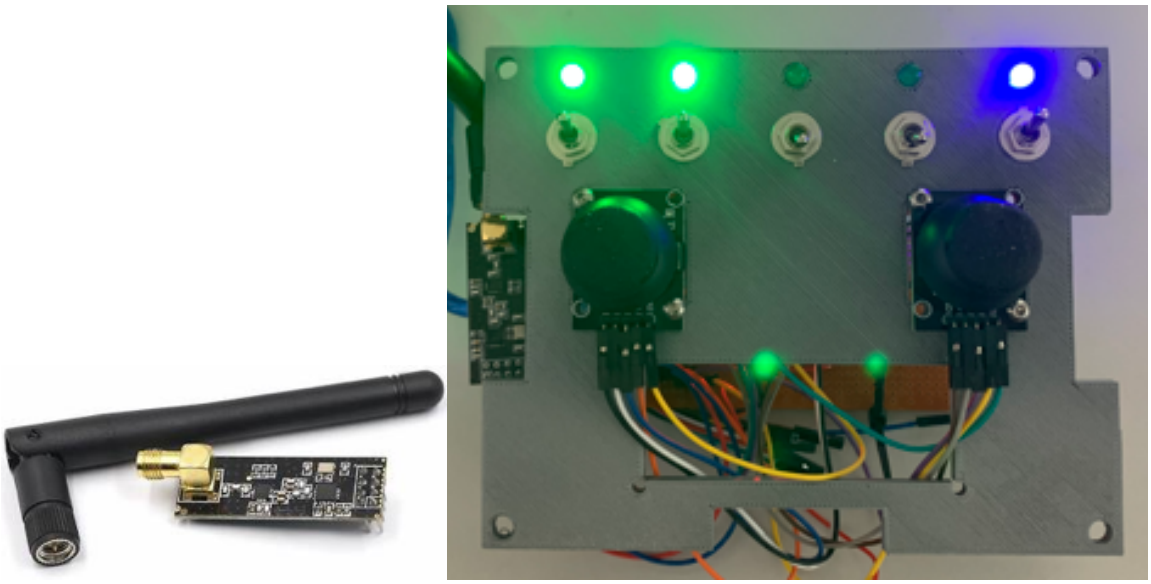

Figure 3-23 Left: nRF24 2.4 GHz radio transceiver module. Right: prototype of the RC controller

The new approach was of a more convenient type, as it was decided to purchase off-the-shelf RC system. As the quality and high resolution of the output was required, the FS-TH9X 9 channel, 2.4GHz radio and the corresponding FS-R9B receiver were chosen. This approach allowed for much more stable connection with a higher resolution signal, which increases both the operational safety of the robot and the control precision. The radio system is presented in Figure 3-24

Figure 3-24 Radio control system; Left: FS-TH9X radio. Right: FS-R9B receiver. (Morele n.d.)

3.4.4. Electrical circuit

Figure 3-25 shows the overall electrical circuit of the robot with all of the previously mentioned components.

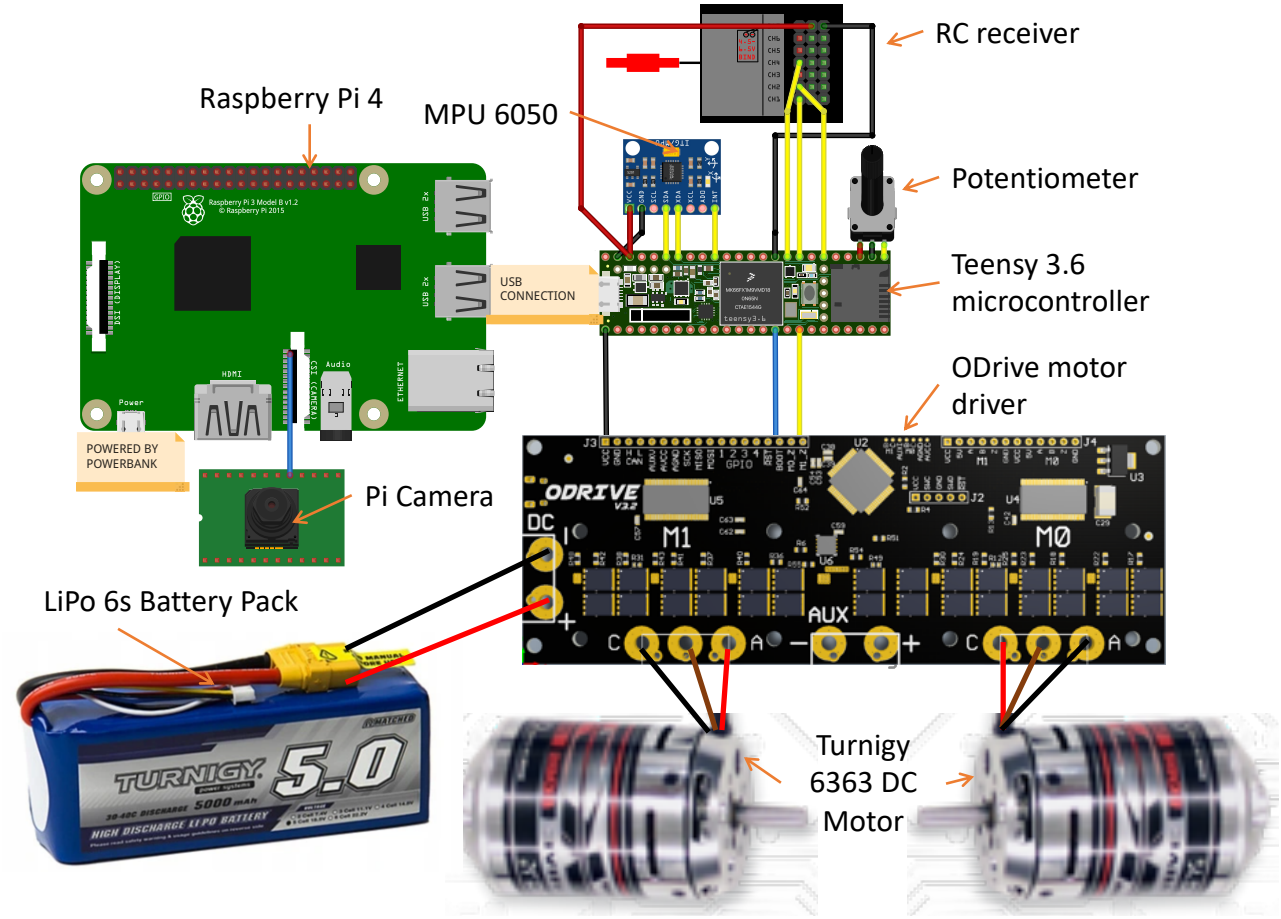

Figure 3-25: Robot's electrical circuit with annotated elements

3.5. Proportional-Integral-Derivative controller

It was indicated in the Literature review section that the State Space controller has the potential to yield the best quality results in the task of dynamic balancing. However, it requires vast amount of knowledge from the field of control engineering that the author of this project simply did not possess and the learning time require in order to construct the appropriate state space controller wouldn't allow to finish the project within the set time frame. Therefore, it was decided to implement Proportional-Integral-Derivative (PID) controller instead due to familiarity with that system and a vast amount of resources available regarding the implementation, tuning and coding.

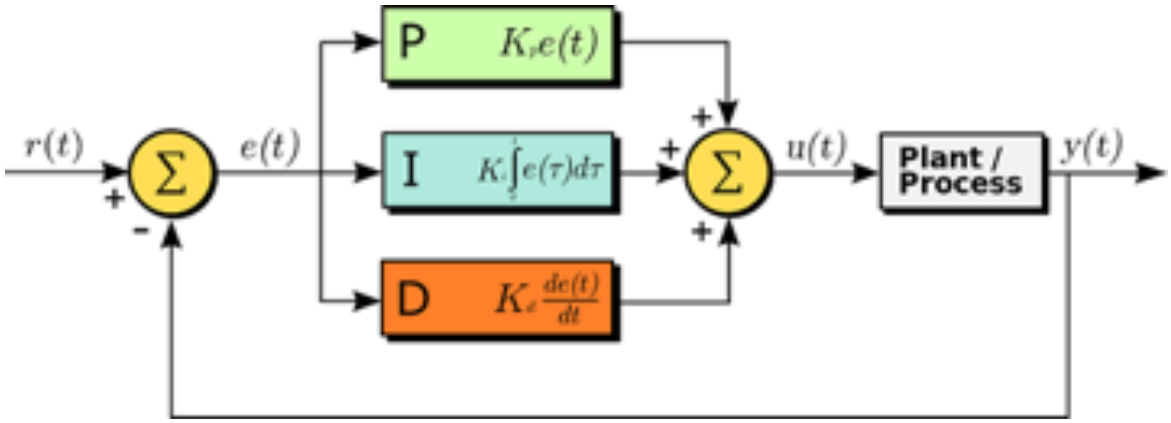

Figure 3-26 Schematics of a closed-loop feedback system with Proportional-Integral-Derivative (PID) controller (Schneider Electric 2019)

The PID controller is one of the most commonly used types of controllers. It is also widely used in self-balancing robots, as it was implemented by Juang and Lurrr (2013), Rahman et al. (2016) and Zimit et al. (2018) in their self-balancing robot projects. The controller operates on the basis of calculating the error (e) between reference (r) and outputted (y) value. The controller consists of three parts, the proportionate that computes control action based on the present error, the integral that computes the action based on the total sum of the error and the derivative that computes the control action based on the predicted future error (Schneider Electric 2019). The magnitude of each of the parts is controlled by respective gains. The influence of those gains is detailed in Table 3-4 (Alsammak 2018)

Table 3-4 Influence of particular PID gains on the controller behaviour (Alsammak 2018)

Parameters	Rise Time	Overshoot	Settling Time	Steady State Error	Stability
K_p	Decrease	Increase	Small Change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Small Decrease	Decrease	Decrease	No influence	Improve

The action of each of the three PID components is calculated based on the respective term from (1)

$$u(t) = K_p\, e(t) + K_i \int e(t)dt + K_d \frac{de}{dt} \tag{1}$$

However, as the robotic system is operating within digital, not analog, domain, (1) needs to be converted from continuous form into discrete form. It is done using the backwards (2) and forwards (3) time shift operators along with the backward Euler and trapezoidal rule approximations.

$$z^{-1}e(t) = e(t-1) \tag{2}$$

$$ze(t) = e(t+1) \tag{3}$$

The backwards Euler approximation is given by (4) below. After multiplying both sides of it by e(t) and substituting the respective term with backward time shift operator, an approximation of derivative between two samples is derived, as showed by (5)

$$s = \frac{1 - z^{-1}}{T_s} \tag{4}$$

$$\frac{de}{dt} \cong \frac{e(t) - e(t-1)}{T_S} \tag{5}$$

Trapezoidal rule approximation gives the integral of the system error as equal to the area between two samples, as given by (6)

$$\frac{(e(t) + e(t-1))}{2} T_s \tag{6}$$

Taking more previous samples into account gives (7)

$$\frac{(e(t) + e(t-1))}{2} T_s \; + \; \frac{(e(t-1) + e(t-2))}{2} T_s \tag{7}$$

Applying the same procedure to all previous samples gives (8). That can be rearranged using $\text{Previous}_{\text{INT}}$ for storing all the previous error, giving the final integral approximation shown in (9)

$$Area_{TOTAL} \cong \frac{T_s}{2}\big(e(t) + 2e(t-1) + 2e(t-2) + 2e(t-3) + \cdots + \; + 2e(t-n)\big) \tag{8}$$

$$\int e(t) \cong \frac{T_s}{2}[e(t) + 2e(t-1) + e(t-2)] + Previous_{INT} \quad (9)$$

Finally, taking the continuous PID equation (1) and substituting respective terms (5) and (9), we get the discretized PID controller equation, visible in (10)

$$u(t) = K_p e(t) + K_i \frac{T_s}{2}[e(t) + 2e(t-1) + e(t-2)] + Previous_{INT} + K_d \frac{e(t) - e(t-1)}{T_S} \quad (10)$$

Equation (10) was implemented in C code form in order to compute an appropriate dynamical balance action. The details of implementing PID are described in next section

3.5.1. Self-balancing PID

The schematics of balancing PID controller system are shown in Figure 3-27 Self-balancing PID controller with system components

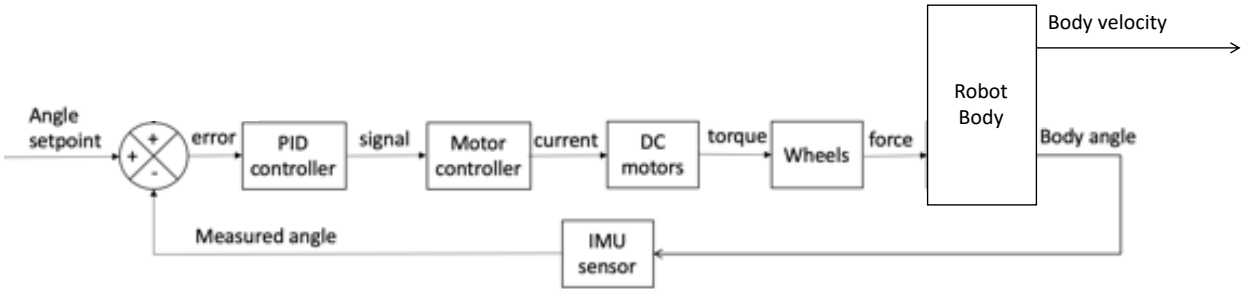

Figure 3-27 Self-balancing PID controller with system components

The PID controller takes the angular position of the robot coming from the IMU sensor as an input and subtracts it from the set vertical position. Then it uses the error value to compute the required control action, which is sent via UART communication protocol to the ODrive motor controller. ODrive sends the corresponding current to the DC motors that accelerates the wheels through the drivetrain system, which in turn exerts an inertial force on the robot's body and thus adjusts its tilt. At that point the loop closes as the body angle is measured by the IMU, subtracted at the summing junction and the whole process repeats continuously.

However, the self-balancing PID controller presents some side effects . Namely, the force exerted on the robot's body, primarily aimed at keeping its tilt at a set value, also causes acceleration of the whole system and a successive gain in velocity. That effect can be easily used to control the velocity of the robot, the process of which is detailed in the next section.

3.5.2. Velocity control

The control of velocity is executed by changing the tilt angle setpoint of the self-balancing controller, which causes the robot to tilt in a given direction. In that position robot can be balanced only if a constant acceleration is applied to it, which is counteracting the gravitational pull. This acceleration is causing change of velocity, which can be measured visually for example by the operator. In that case the operator performs the function of controller, as he adjusts the set tilt angle using RC pilot. In that way the operator can cause the robot to accelerate or deaccelerate, thus positioning the robot as it suits.

In the same way the autonomous control system can use the velocity control to position the robot accordingly. If detected person is too far away, the robot needs to accelerate for a given period of time in a given direction, in order to decrease the distance between itself and the followed person.

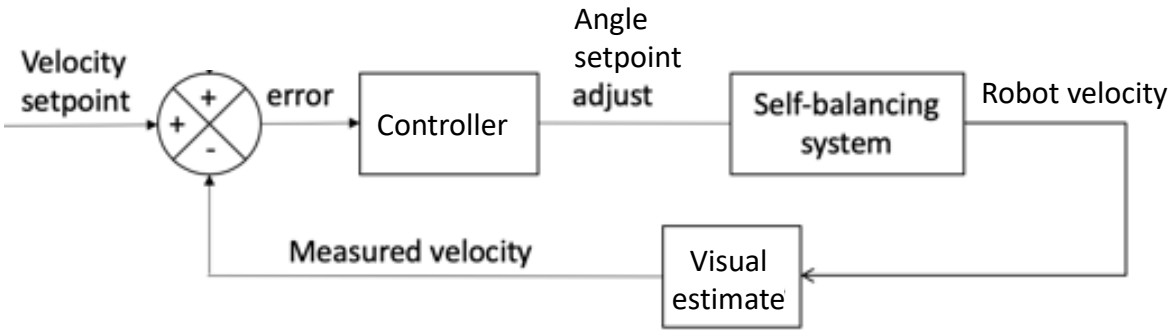

Figure 3-28 Velocity control feedback loop

3.5.3. Direction controller

The direction of the robot's heading is adjusted using simple open-loop, proportionate control. The controller receives signal from the self-balancing PID and from the RC system or the autonomous control system. If there's no signal coming from the radio or autonomous control, the controller is omitted. Once the signal is inputted, it is scaled proportionally and the drive current value sent to wheels is adjusted accordingly either by addition or subtraction of the input signal value. For example, if the robot is to turn left, the turning value would have to subtracted from the left wheel and added to the right wheel. The schematics of heading direction control are shown in Figure 3-29.

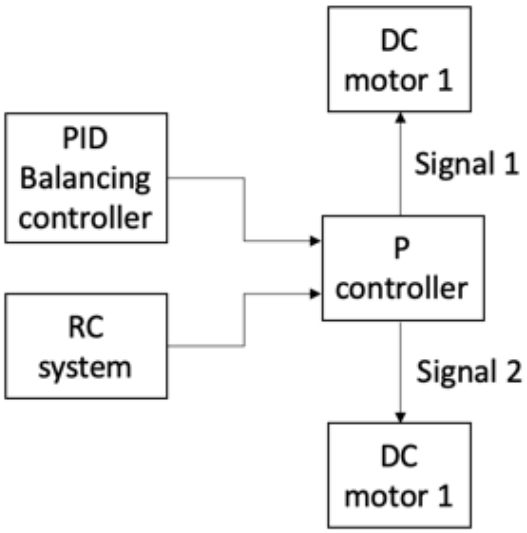

Figure 3-29 Proportionate direction controller

3.6. Computer Vision design

As stated in Section 3.1.2.3, the robot is ought to have visual perception capabilities that would allow it to possess spatial awareness and be able to recognise and follow a given person.

Computer Vision techniques vary in their characteristics and designs. In case of this robot, the goal was to create a vision system that would have its inner working similar to what humans are doing in order to locate people and objects around us and gain spatial and distance information about our position in relation to the immediate environment. It was decided to have only one environmental information input – camera feed. The hypothesis was that this one stream of data should be satisfactory to achieve the goal of autonomous navigation. Section 3.6.2 details the software architecture created in order to achieve that goal with deployed hardwired described in Section 3.6.1

3.6.1. Hardware

Computer vision tasks are usually highly computationally intensive and couldn't be realistically performed on a microcontroller, thus a designated hardware system was deployed . The heart of the CV hardware lays in a small portable single-board computer Raspberry Pi 4. The computer is running Raspbian OS and allows the Python environment to be used. The camera feed is provided using the Wide-angle version of Pi Camera, which is equipped with 8 Megapixel camera sensor providing satisfactory image quality.

Object detection is very computationally intensive task that usually requires dedicated GPU system in order to be run in real time. In order to provide real-time computer vision functionality to the raspberry pi, the Intel Neural Stick 2 was used. It is a dedicated GPU-like mobile processor that allows to efficiently run AI application on the edge devices. All components can be seen in Figure 3-30

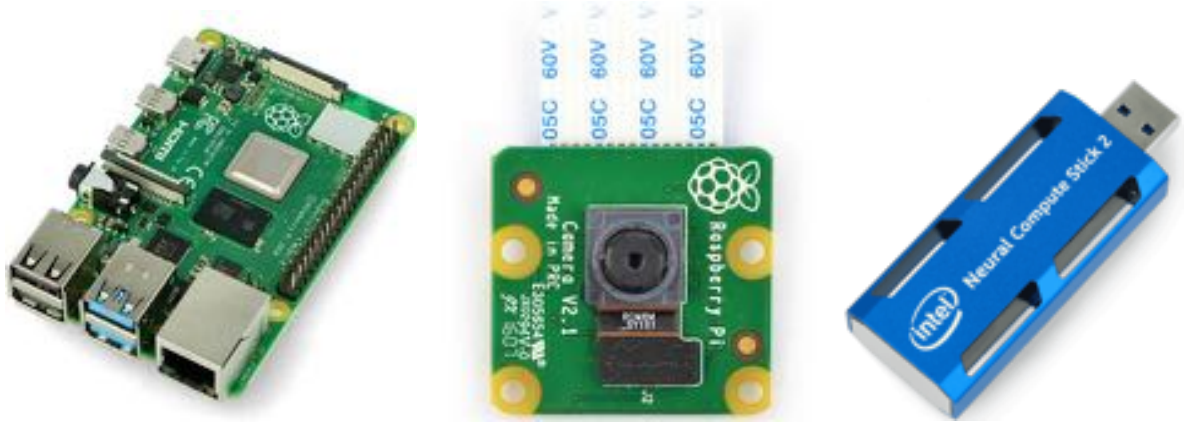

Figure 3-30 Computer vision hardware. Left: Raspberry Pi 4. Middle: Pi Camera Module. Right: Intel Neural Compute Stick 2 (Botland n.d.)

3.6.2. Software architecture

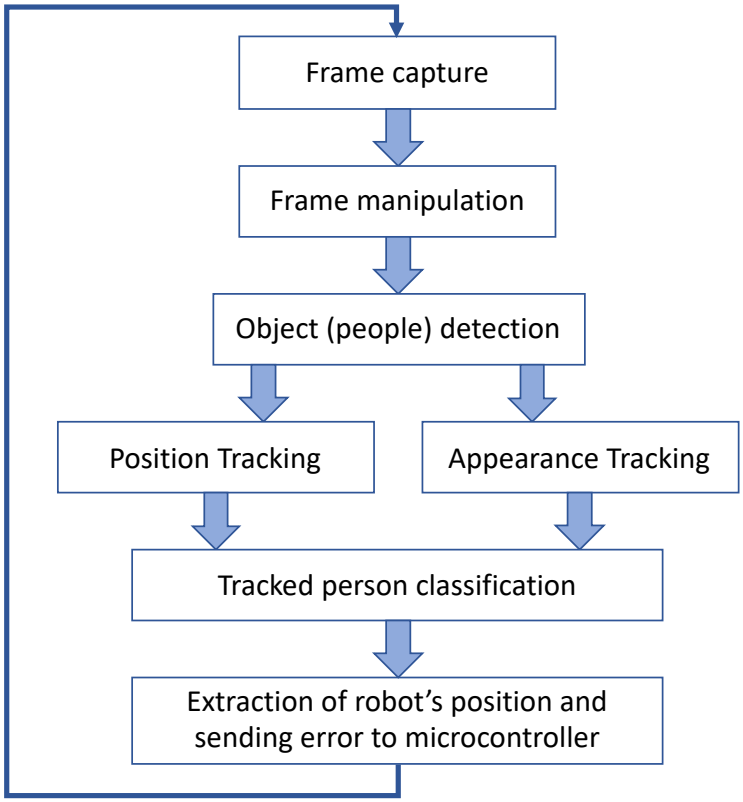

Figure 3-31 Robot's vision system architecture

Figure 3-31 shows a diagram detailing the looped instruction together creating the computer vision system. The algorithm is set as follows:

0. Before starting the loop, run short training procedure in order to extract the current position and appearance data of the person to be followed. That step is run only once
1. Start the loop. Capture new frame using onboard camera
2. Manipulate the frame in order to set it to a format accepted by the object detector
3. Perform object detection in the frame using convolutional neural network object detector
4. The outputs of detection are likely to contain few different people
   4.1. Extract the centroids of detected people and compare their locations to the last known location of the followed person
   4.2. Extract appearances of the detected people and compare this data to the last known appearance of the followed person
5. Based on the position and appearance tracking, decide which person to follow
6. Extract the robot's position in relation to the followed person
7. Compute the error between the desired and current position
8. Send the position error to the microcontroller
9. End the loop, go back to step 1

3.6.3. Frame capture and manipulation

The data input to the system is coming from the camera sensor. This data needs to be interpreted correctly and formatted accordingly to the object detector requirements. To perform those actions, OpenCV library is used.

OpenCV is an open-source library of programming functions mainly aimed at real-time computer vision. It is highly optimized with an intent to be used in real-time application (opencv.org 2020). The library is used to:

1. Capture the frame from webcam stream – see Figure 3-32
2. Resize the frame to speed up the later computation
3. Create blob from image. Creating a blob involves scaling the image and mean subtraction of the Red Green Blue (RGB) values, in order to supply the object detector with an image of appropriate format

Figure 3-32 Captured frame of resolution 1080x720p

3.6.4. Object detection

Once the frame is captured and properly formatted, then object detection can be performed on it. It is done by utilisation of MobileNet Single Shot Detector. Object detection is generally based on using convolutional neural networks, however in majority of detectors two stage approach is taken: firstly probable objects are detected within the frame and secondly classifiers are run on those areas to classify objects into the classes. MobileNet SSD is doing both of those task in one shot, thus its name. Thanks to that architecture it can perform object detection much faster than the traditional tools without losing much of accuracy, therefore it is a great solution for mobile computing projects such as this one. Figure 3-33 shows the example of an output from object detections. It can be seen that

detected objects, in this example two people, are enclosed using bounding boxes. For each of the detection a probability and class are given. Probability means the confidence that detector has that the given object is there and belongs to the denoted class.

Figure 3-33 MobileNetSSD object detector output showing detected people with annotated confidence. Resolution: 400x225p

3.6.5. Person tracking

As seen in Figure 3-33, in an example frame two people were detected. The detector doesn't differentiate between them, as its function is limited solely to detection of any person in a picture. However, as the robot should follow only a given person during its operation, a need arises for a tool that would differentiate detected people and keep track of who is followed across frames.

The solution proposed here consists of two methods that in combination should prove sufficient in performing given person tracking. The tracker is divided into two components:

- Position tracking
- Appearance tracking

Both methods are detailed in sections 3.6.5.1and 3.6.5.2

3.6.5.1. Position tracking

Position tracking, or precisely centroid tracking, is one of the most basic and most useful tracking methods. In case of this project, its structure can be explained by the following looped steps:

1. If there currently aren't any followed or tracked people, run object detector and take the person closest to the centre of the frame as the followed target
2. Run object detector
3. If there weren't any detections, keep on executing step 0 till a person detection happens
4. Extract the positions, or centroids, of all detected people
5. Calculate Euclidian distances between the newly acquired centroids and the previous knows position of the followed person
6. The centroid that is located closest to the previous known position is taken as the new, or updated, position of the followed person
7. Loop back to step 1

The steps outlined above are described visually on the example where the previous position of the followed person was knows, as shown in Figure 3-34.

The sequence shown in the mentioned figure also indicates some inherent issues of position tracking. In the last frame of the sequence the tracked person has disappeared and the algorithm has taken the other person present in the frame as the one to be followed, as its distance was the smallest to the previous known position of the followed person. Therefore, position tracking is a solid basis for a tracking algorithm, however it requires to be coupled with an additional component that would take into account some visual features of a tracked object.

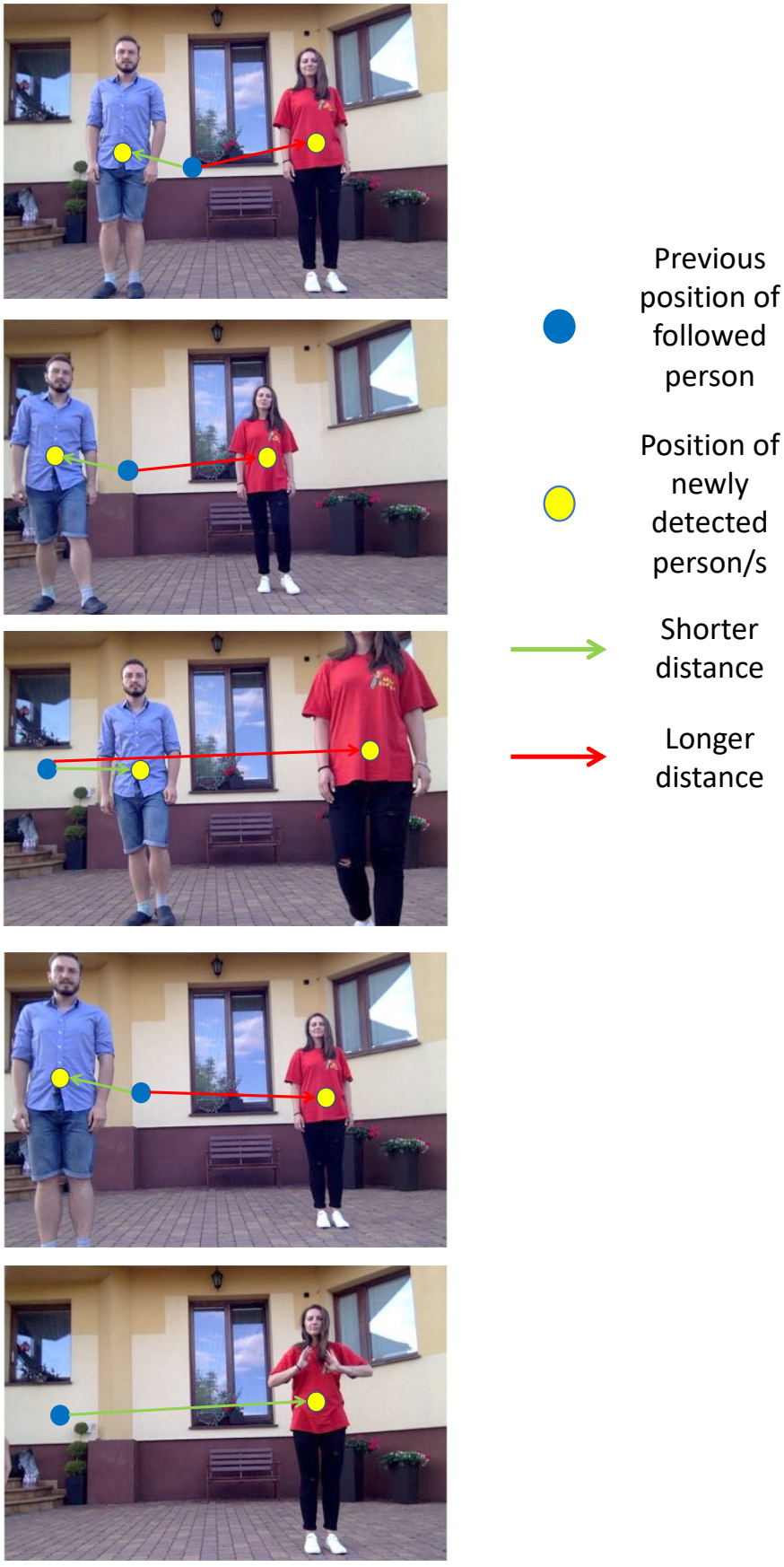

Figure 3-34 Visualization of centroid tracking with a legend. The updated position of followed person is determined as the closest newly detected person position in relation to the previous position.

3.6.5.2. Appearance tracking

As mentioned in 3.6.5.1, position tracking alone might not be sufficient for object tracking algorithm as occlusions, path-crossing or misdetection might happen. In order to complement the tracking system, it was decided to design and implement an appearance tracking algorithm that would be able to extract visual information about detected people.

The approach taken here is based on extracting colour information from within the bounding boxes that usually are drawn around detected objects, as shown in Figure 3-33.

Colours can be mathematically represented using colour models. Colour model is an abstract mathematical model describing the way colours can be represented as tuples of number, typically as three or four values or colour components (Wikipedia 2020). There are numerous colour models, with one of the most popular ones being Red Green Blue (RGB). The RGB colour model is an additive colour model in which red, green and blue light are added together in various ways to reproduce a broad array of colours (Wikipedia 2020). The additive nature of RGB colour representation is depicted on the left side of Figure 3-35. RGB is widely used in representation and display of images in electronic systems, however for applications within computer vision it is not the most applicable model. It is so because lightning conditions frequently change in the real life environment and even slight variation in how the scene is lighted can have a major impact of how the scene's colours are described using RGB. Therefore, an alternative approach of using Hue, Saturation and Value (HSV) colour model is used. HSV is an alternative representation of the RGB colour model, designed in the 1970s by computer graphics researchers to more closely align with the way human vision perceives colour-making attributes (Wikipedia 2020). In this model colours of each hue are arranged in a radial slice with the saturation dimension resembling various tints of brightly coloured paint and the value dimension resembling the mixture of those paints with varying amounts of black or white paint, as presented on the right side of Figure 3-35.

The advantage of HSV model RGB in computer vision applications is that the hue of surfaces or objects doesn't change regardless of the lighting conditions, with only the saturation and value values being sensitive to those conditions. That means that theoretically one could track appearance of a given hue across a range of scenes with varying lighting conditions, which is a close approximation to the real world conditions. Therefore, hue-information extraction and tracking was taken, as explained below.

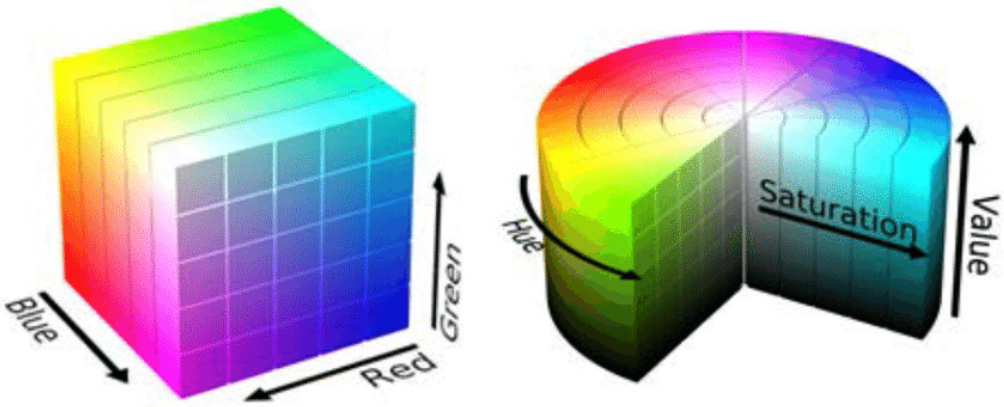

Figure 3-35 Red, Green and Blue (RGB) colour model (left) and Hue, Saturation and Value (HSV) colour model (right). (Roza et al 2016)

In the appearance tracking algorithm we are interested in extracting colour information from the image area in which detected people are located. That area is encompassed by the bounding boxes, thus the exact pixels building the image of detected person can be isolated. Each of those pixels contains three values, which in OpenCV take the following value ranges:

- Hue: 0 – 179
- Saturation: 0 – 255
- Value: 0 – 255

However, as the saturation and value dimensions can be disregarded, we only need to work with hue values, meaning that for each pixel we have one value in range of 0 to 179 representing its colour information. Thinking in terms of an surface area, our appearance is mostly dictated by the clothes we wear, which in turn tend to have rather consistent, single or few coloured structure. That means that in theory the colours, or hues, should be grouped in form of clusters. Each of those clusters represents the amount of a given colour or small range of colour values that is present in our image representation.

There are a few methods of finding data clusters, with one of the most popular ones being unsupervised machine learning algorithm called k-means clustering. *k*-means clustering is a method of vector quantization that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean distance between the observation and the cluster centre (aiming at minimization of squared error) (Wikipedia 2020). The application of k-mean clustering method on the pixels hue data results in creating a number of clusters with the central hue value serving as a cluster centre or the most representative hue value for the neighboured hue values. In other words, the more values are associated with a given cluster centre, the more representative that hue value is of a proportionally big part of the area's colour information.

The properties described in the previous paragraph can be used to extract colour information from an area of interest and cross compare that information with another area, thus being able to differentiate the two areas based on the visual appearance information solely. The magnitude of differences in the appearance data is proportional to the variety of colours that are worn by the detected people. To visualise the potential difference in appearance data, the areas of detection from Figure 3-33 were pipelined through the appearance extraction algorithm with the respective results being presented in Figure 3-36 and Figure 3-37 with a settings of extracting both 8 and 1 cluster centre/s.

It can be seen that for both 1 and 8 cluster centres the differences in appearance data are quite pronounced. Comparing the 8 cluster centres data, for the left-side male the majority of hue values lied in the hue value area of 0 -20 and some were scattered across the range of 110 to 170. For the right-side female there was a spike of amount of pixels around a hue value of 130, with some further clusters placed in the range between 140 and 180/0. This overall shift trend was confirmed by the single cluster centre data, which was set to be around the value of 60 for male and 120 for female.

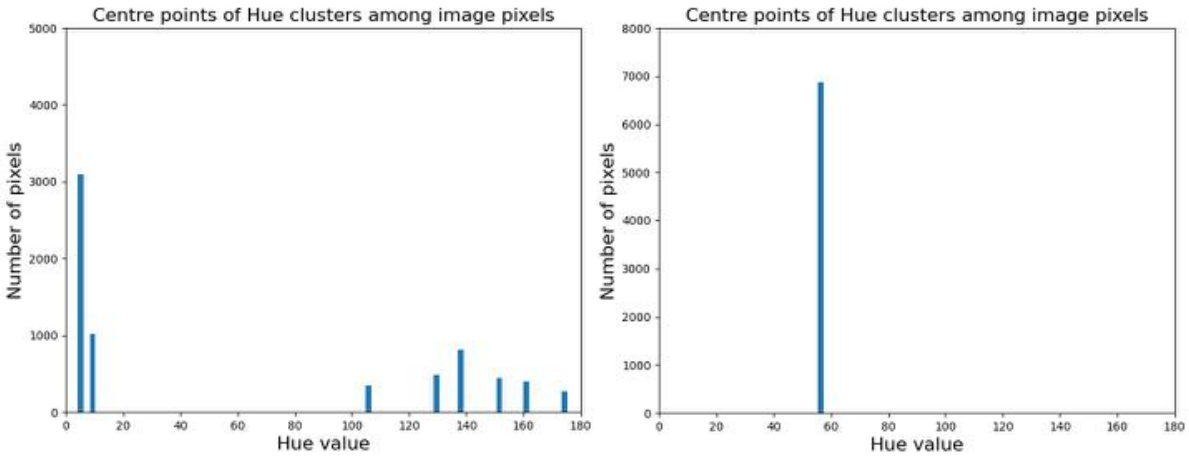

Figure 3-36 Appearance data for left-sided male from Figure 35

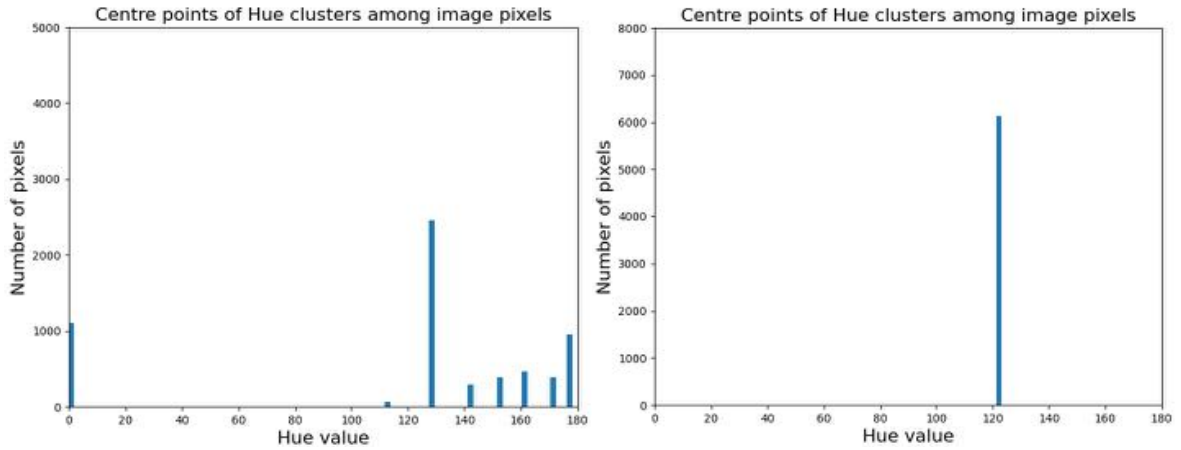

Figure 3-37 Appearance data for the right-sided female from Figure 35

Building on the supplied information, the following approach was taken for the appearance tracking algorithm:

1. Divide the captured frame into areas marked by the object detector's bounding boxes likely containing detected people in majority of its area
2. Change the colour model of the supplied images from RGB to HSV in order to be lighting-insensitive
3. Extract hue (H) value of each pixel from the image
4. Run k-means clustering algorithm on the pixels hue data
5. Retrieve cluster centres (most representative hue values) with the count of belonged pixels
6. Compare the extracted colour information with the known ground truth appearance data of the followed person by method of:
    6.1. Dividing the hue range into sections representative of real life frequently existing colours spectrums and comparing the mount of pixels in those ranges
    6.2. Attaining the single cluster centre for a given image and comparing its location

3.6.5.3. Tracked person classification

Overall, the tracking system consists of two parts: position tracking and appearance tracking. The way the two subsystems are interplaying is:

1. The main tracking components is the position tracker due to its low computation cost and acceptable short-term performance.
2. The appearance tracker serves as complementary part. It is magnitudes more computationally expensive to run, therefore it is called every few frames in order to check whether the position tracker hasn't made any mistake.
3. The appearance data is compared to the ground truth data gathered at the start of the following procedure (see 0. In 3.6.2). Additionally, all the entries that are accepted as comparable with the ground truth data are added to the appearance data pool, creating running average data values. That approach helps to adjust to the inevitable changes in the person appearance due to background and differences in lightning

3.6.6. Extraction of robot position

Once the detections were performed and the person of interest was filtered and set to follow, the robot needs to knows its position in relation to that person in order to execute adjusting action. The robot's position is calculated two-fold:

- The vertical position is taken relative to the vertical position of the detections bounding box coordinate, with an aim of overlap between the horizontal value of frame centre and the followed person centroid
- The distance to the person is calculated using the height of the bounding box. It is set that the bounding box should have the height of 80% of the frame height, which relates to a certain position between the robot and the followed person. This measurement is not exact, however the robot only needs relative, not absolute value. That approach can only work is it is assumed that the whole person is enclosed by the bounding box.

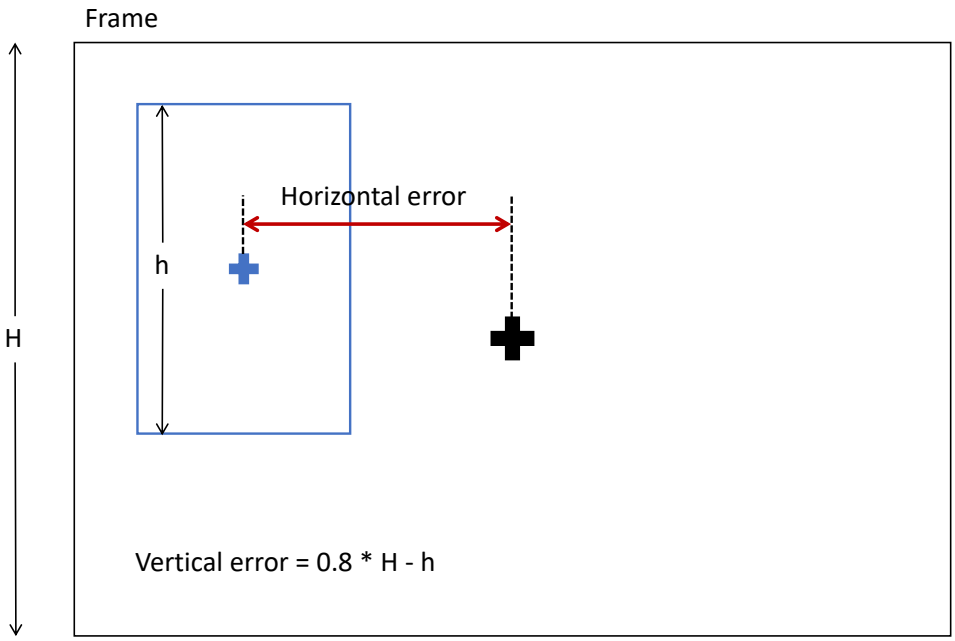

Figure 3-38 The method of calculating the distance and direction errors of the robot-followed person relative system

3.6.7. Communication with microcontroller

The positional error is send to the robot's microcontroller using Serial communication between the Raspberry Pi and teensy via usb ports.

3.7. Autonomous behaviour

The autonomous behaviour of the robot is based on the concept of data interpretation and taking appropriate action. Therefore, the behavioural algorithm is a bridge between the positional output of the computer vision system and the position manipulation balancing control system.

The behavioural algorithm was built using a decision tree logic which outlines what steps shall be taken by the robot in a response to a given input of position data. That approach allows the robot to seem and behave like an intelligent entity capable of reasoning and cognitional activity. In reality, this intelligent behavioural is just a combination of a number of simple logic gates and thresholds, which may be understood as simplistic, however it is a very good approximation of how humans reason.

Taking an example of giving someone an order to follow a given person, that task can be broken down into a number of subtasks, which then can be broken down further till we arrive at the most simplistic logic gates, similarly to what is happening in the robot's behavioural algorithm. A loose structure of such task would be put hierarchal like:

1. Recognize the person in the immediate surroundings
2. Approximate the distance and angle between your body and the followed person
3. Actively try to adjust those values using muscle systems so that they are in the range that your sense would deem reasonable

The robot's behavioural algorithm takes form very similar to the steps outlined above, with a distinction of having an exact, pre-set values for thresholds, distances and angles, which were determined using common sense and trial and error. The schematics of the algorithm in the form of decision tree can be seen in Figure 3-39.

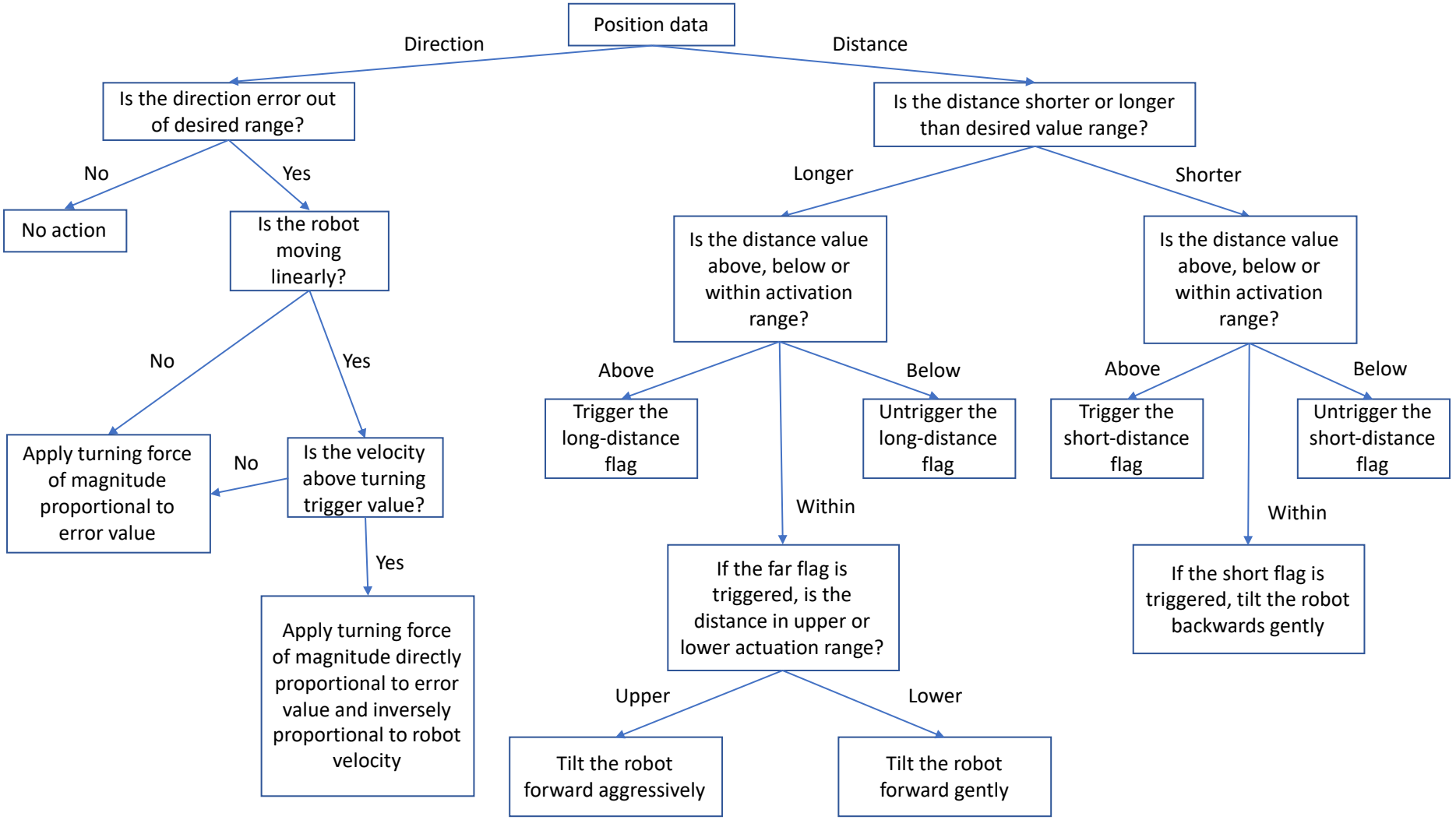

Figure 3-39 Decision tree of the autonomous navigation algorith

3.8. Code

All the hardware was programmed by the author of this project with help and use of available libraries and online tutorials. For the Arduino and Teensy, C language was used within the Arduino IDE to connect all the peripherals, manipulate the data and send appropriate actuation signals. For the Raspberry Pi and computer vision (CV) system, Python language was used to manipulate to create the structure of the CV system and communicate with Teensy.

Full Teensy code is available in 6.1 Appendix 3 - Microcontroller code (C)

4. Results

This section contains the results of gathered across a range of tests and verifications. The respective data is shown and explanation for phenomena is given.

4.1. PID tuning

In order for the PID controller to work properly, its proportional (Kp), integral (Ki) and derivative (Kd) gains, which are responsible for regulating the strength of each component, need to be tuned properly. The controller can be tuned twofold, either by model-based or trial-and-error approach. The first approach is based on creating a model of the system, using transfer functions or state-space representation, and placing poles and zeros of the controller, which are factors regulating its activity, into desired positions along root locus. Those positions will then yield an appropriate gain values that can be interpreted in code and implemented int the controller. That approach allows to fine-tune the working of the controller so that the most optimal performance can be achieved. However, creating a model of the system can be challenging. Even though there are supporting methods, like system identification, as the complexity of the system is rising, so does the difficulty of modelling it and the accuracy of the model decreases. The mechanics of self-balancing robot, as indicated in Section 2.1.1 Mathematical model and control, are very complicated with non-linear equations of motion and would require considerable amount of time to be modelled accurately with all existing friction losses and exact weight placement. Therefore, the second approach of controller tuning was taken, namely trial-and-error.

The trial and error approach is based on understanding the working principles of PID controller, the influence of each of the terms and ability to make valid prediction of how changing the gain values will impact the system. In order to tune the controller, the guidance of terms influence presented in Table 3-4 was used coupled with engineering common sense. The process is outlined below.

Proportional gain (Kp) lays at the heart of each PID controller and the tuning process was initiated with checking the influence of this term on the robot behaviour, with results visible in Figure 4-1. Firstly, direct proportionality between the pitch angle and set motor current was tested (Kp = 1). That value prevented the robot from immediately falling over, however it was found that system was unstable, as it was oscillating rapidly with ever increasing angle amplitude. Next, the Kp value was increased to 2, however that only caused the robot to reach instability even faster. The instability could be caused by over actuation, or too high gain, therefore the Kp value was reduced to levels of 0.75 and 0.5. For the first value the oscillatory behaviour was still present with slightly slower amplitude increases. For the second value the system was oscillating even slower, however the robot

was also gathering considerable amounts of speed alongside the actuation. At this point it was clear that the system cannot be stabilised only by proportional term and it was decided to keep the Kp value of 0.5 and try to increase the actuation with introduction of integral gain.

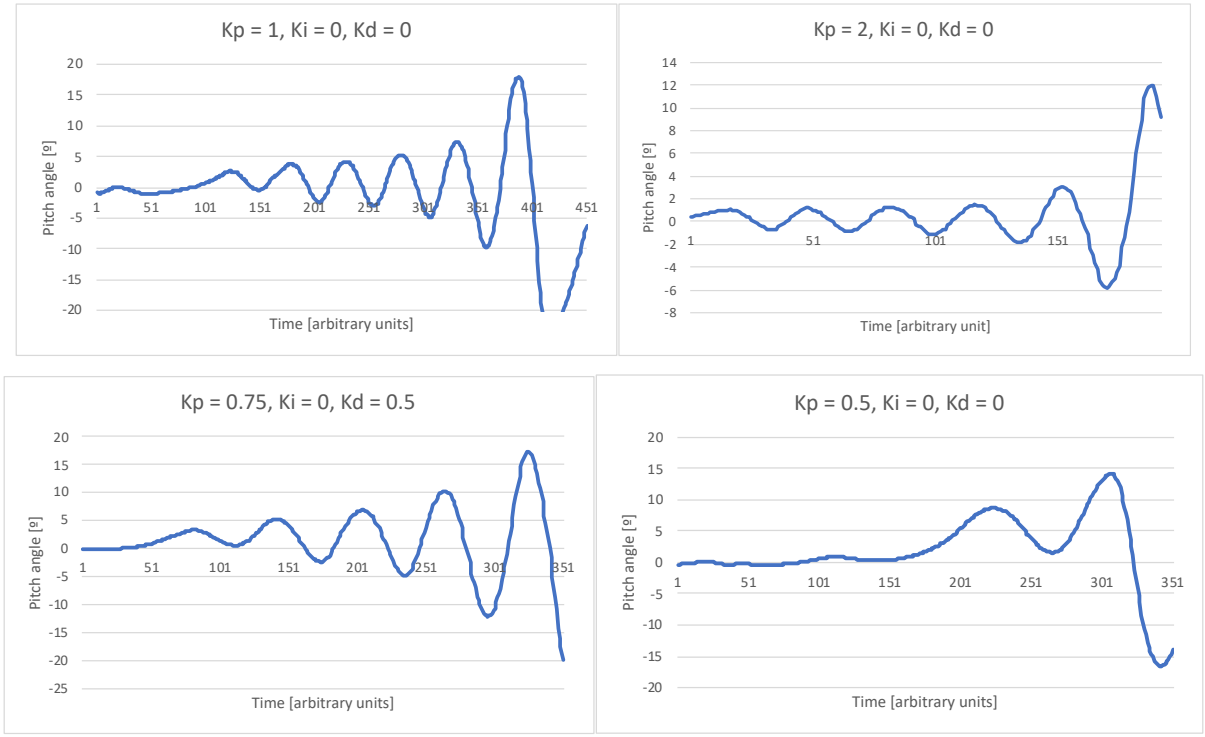

Figure 4-1 PID tuning: proportional gain. Values from 2 (upper-left), through 1 (upper-right) and 0.75 (bottom-left) to 0.5 (bottom-right) were tested

As the Kp values were tested, it could be seen that the system had problems with providing proper actuation as the wheels were driven with notable force only once the robot's pitch has surpassed a relatively large angle. That meant that the theoretical stabilisation was only possible if the system was to oscillate between large angles, of magnitude in the range of 15 degrees. Therefore, it was deducted that the actuation must happen earlier, before system reaches critical angles. Thus, integral gain (Ki) was added on top of Kp = 0.5 as presented in Figure 4-2. The first tested value of Ki was 0.01, however it still left the system reaching extreme angles and go unstable. As a matter of guess, the Ki value was increased to 0.03, which still caused the system to reach rather extreme angles of magnitudes close to 15 degrees. This time, however, the angles weren't increasing with time and it seemed that the system has became marginally stable. Adding more actuation didn't seem like a smart guess as visibly judging, the system was already actuated very strongly. Instead, the so far influence of gains was analysed. Both proportional and integral gains are ignoring the direction in which the system is about to head. The proportional gain causes actuation when the system is already reaching unstable regions. The integral gain is working based on the collected error, therefore it has a potential to start actuating

soon after the pitch angles are reaching meaningful regions. It was deduced that a third component was needed, one that would act as soon as the robot is moving from one side of the pitch to the other.

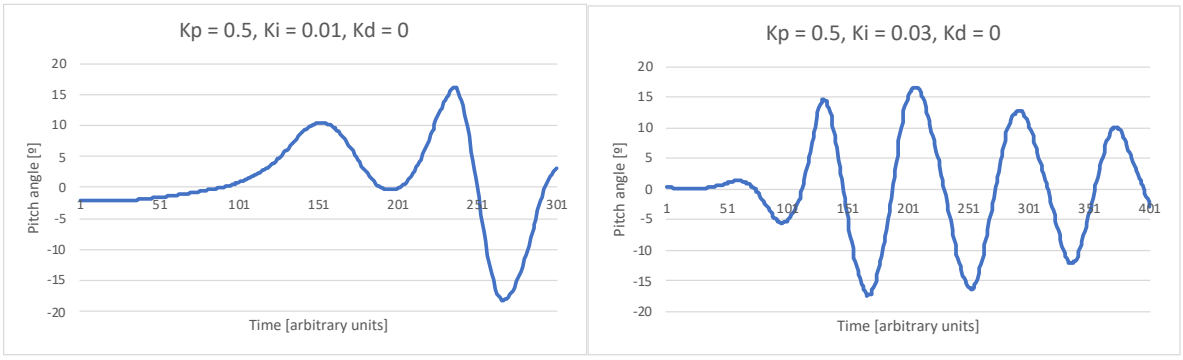

Figure 4-2 PID tuning: integral gain. Values of 0.01 (left) and 0.03 (right) were tested

Therefore, the derivative component was added. It was thought that this component should act as soon as the robot is changing its side of balance and also proportional to how fast that angle is changing – as the faster the change of angle, the stronger counter actuation should be. The derivative gain (Kd) is working exactly like that. The main actuation providing balance would be coming from the integral gain, which also prevents the robot to stay at one of the numerical sides of the pitch angle for too long. The proportional gain would be supporting the integral gain in the actuation and providing a linear, predictable actuation. The parameter that would stop the robot from falling too far towards one of the numerical sides of the pitch angle would be derivative gain. An analogy might be set up here, that if robot's dynamic would be compared to person feeling a pain due to injury, the proportional part would cause action proportional to the pain, being invariant of time, like calling a doctor if the pain gets too severe. The integral gain would be analogical to calling a doctor only if the pain is too severe for too long. The derivative gain, in this analogy, would be not letting the pain to start in the first place. As predicted by the thought experiment, the derivative gain has allowed to dynamically stabilise the system, as presented in Figure 4-3, with the value chosen to be around 100, based on the max current that could be safely put into the motors.

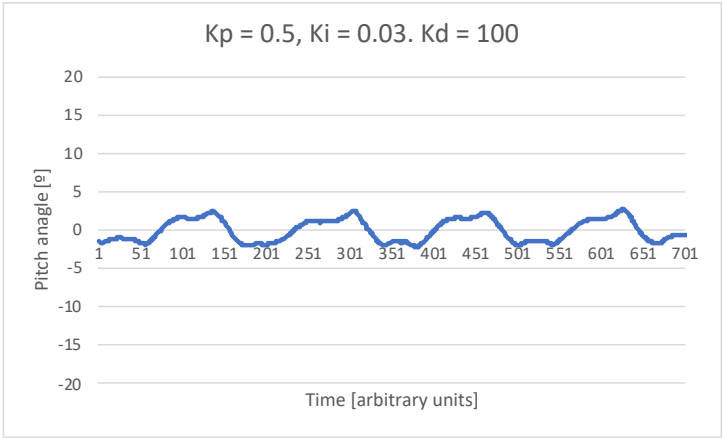

Figure 4-3 PID tuning: derivative gain. Value of 100 was tested

4.2.Self-balancing

During the self-balancing operation, if robot tilts, or change its pitch angle, towards a given side, the wheels are actuated in to order to accelerate in that direction and exert inertial force on the robot counteracting the influence of gravity. In theory such action should allow to precisely place the robot in the balanced position. However, as real world has disturbances and the controller isn't perfect with constant overshoot or under actuation, the dynamical stabilisation is resulting in an oscillation around a balancing point. The good controller design would be characterised by keeping those oscillations to very small angle at a low frequency.

The self-balancing performance of the system presented in this study is showed in Figure 4-4. It can be seen that as the robot falls in one direction, the PID control action is acting in an opposite direction, thus reducing the tilt. However, the controller action is causing a slight overshoot, therefore the robot starts to tilt in the other direction and the cycle repeats. The oscillation is range is relatively small, as it ranges from -2 up to 2 degrees, which results in small, although visible oscillatory movement.

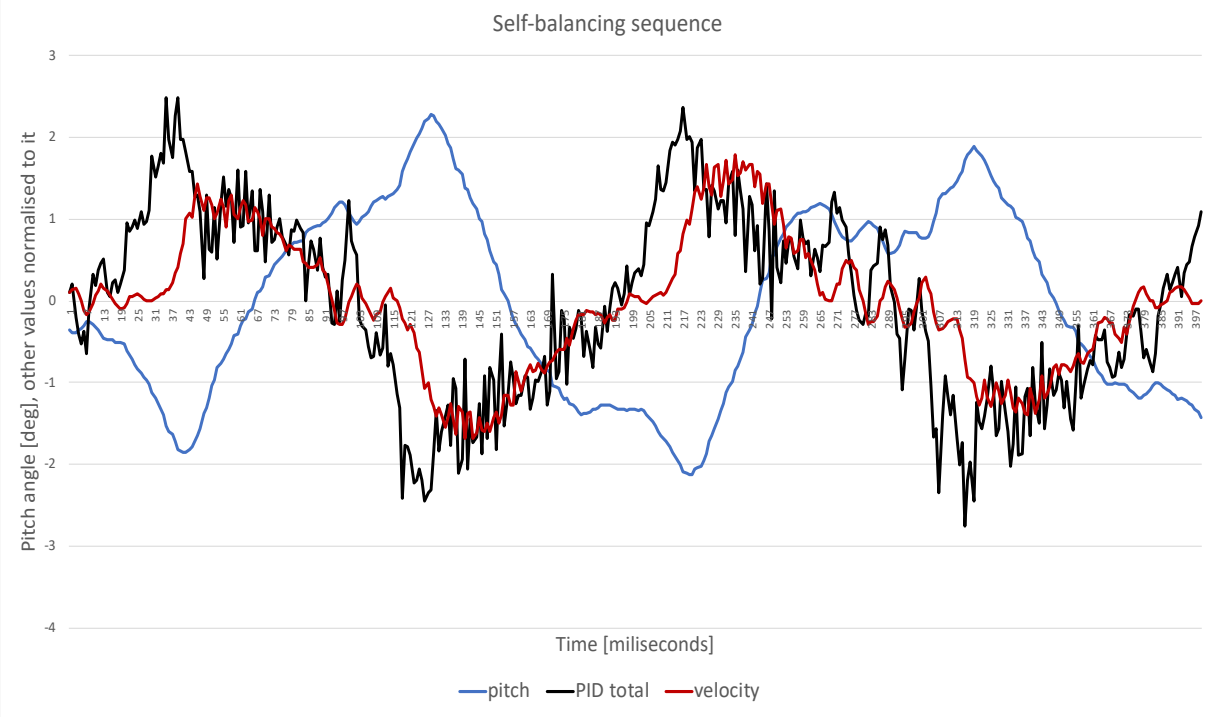

Figure 4-4 Robot self-balancing sequence

Figure 4-5 details the action of each of the PID gains during self-balance sequence. It can be seen that the proportional part is acting as simply opposite to the pitch angle with a scaled value. It would be hard to stabilise the system simply by applying this part, as there's a very narrow range of the correct

gain values. Outside of that range the system would either be underactuated and fall down or over actuated and result in instability with increasing oscillations amplitude. Due to the experimental nature of the tuning, the single proportional gain value couldn't be find. Therefore, the integral part was added. However, it can be noted that the integral action is visibly lagging behind the pitch value, which probably causes the persistent oscillatory behaviour. The lead-lag compensator could be added in other to put the integral action in phase with the pitch angle and stabilise the system further. It was found that the PI controller was producing oscillations of incremental amplitudes inevitably leading to instability. To counteract that behaviour, the derivative components was added. It can be seen that the derivative action is 'predicting' the future state of the system. This is especially visible when the pitch value stops to increase and start to increase. At this point the derivative action is acting almost instantaneously, which helps to reduce the amplitude of the oscillations. In theory, further increase of the derivative gain should result in decrease in the oscillatory amplitude. However, it was found that such an increase caused instability in the system, as the derivative gain was causing a sort of closed loop amplification on itself, as if it tried to counteract its own behaviour. Moreover, it can be seen that the value of derivative action is changing very rapidly with sharp spikes and almost constant oscillation around a leading value. That might be due to insufficient filtering of the IMU data and could be solved by applying moving average filter to it, which should decrease the oscillation of the derivative action.

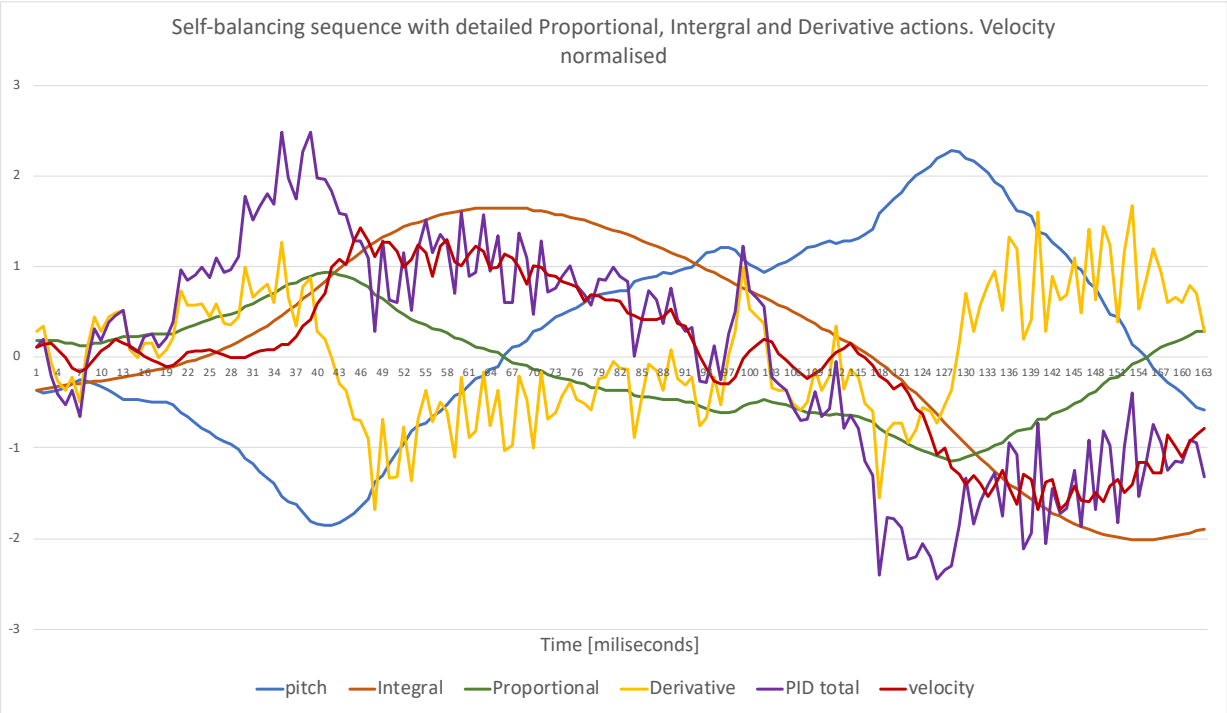

Figure 4-5 Robot self-balancing sequence with distinction of Proportional, Integral and Derivative gains actions

4.3. Radio controlled drive

This section presents the results of radio-controller drive of the robot, which is using the RC pilot signal as angle setpoint thus changing input into the balancing PID controller. The operating principle of such approach is as follows: the angle setpoint is changed so that the robot has to lean in a given direction. As it does, it keeps its tilted position. In order to balance at that position it needs to constantly accelerate in order to produce inertial force that counteracts the acting of gravity pull. This acceleration gradually increase the velocity and once a desired level is reached, user releases the changed setpoint and the robot goes back to the upright position. However, the velocity gained in the process isn't lost. As stated by the Newton's First Law of Motion, 'in an inertial frame of reference, an object either remains at rest or continues to move at a constant velocity, unless acted upon by a force". What that means for the robot is that it doesn't matter whether it is moving or it is stationary, as in both cases there are no external forces acting on it, if the air drag and wheels friction can be neglected. In reality, due to internal resistance of the motors and constant balancing action the robot will gradually slow down, however the velocity can be increased once again by changing the balancing setpoint. Data presented in Figure 4-6 reflects that behaviour. It can be seen that as the angle setpoint is remotely changed, the robot leans in a selected direction and by doing that it gradually increases its velocity. Once the desired velocity is reached, the angle setpoint is changed back to the neutral value. If the user wishes to deaccelerate, the angle setpoint is changed in the opposite direction, which causes the robot to gradually decrease its velocity.

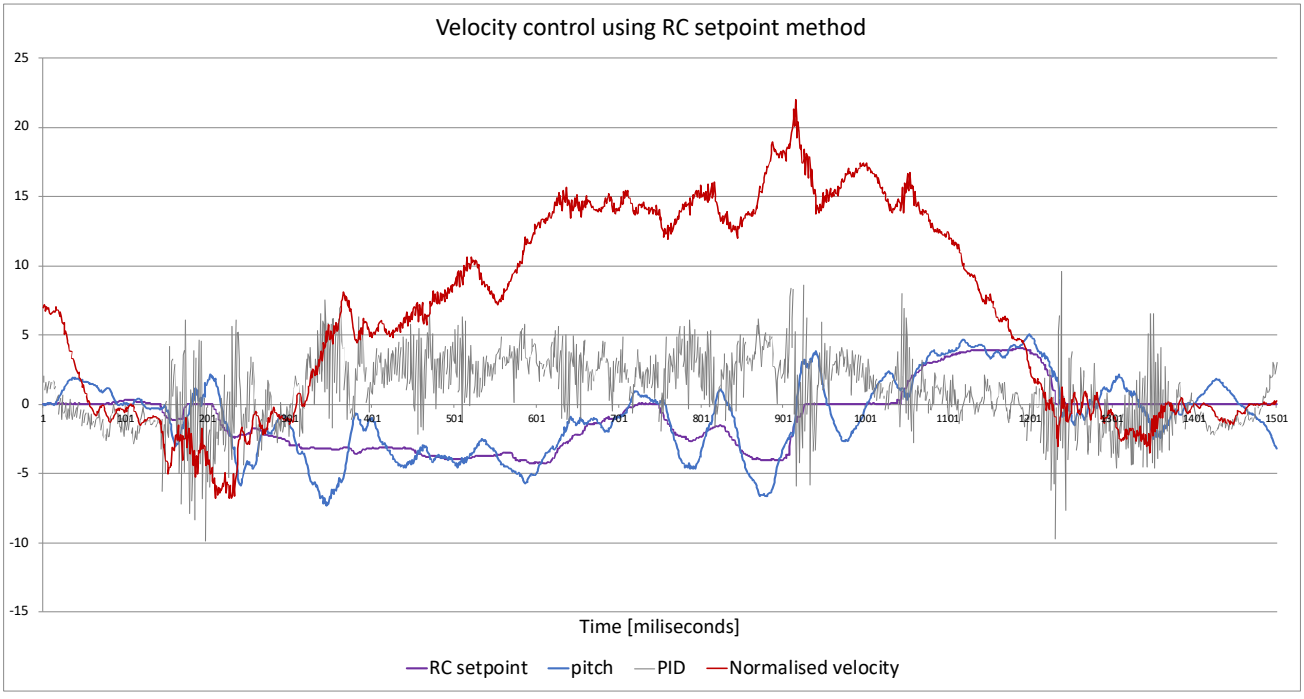

Figure 4-6 Velocity control by utilisation of RC angle setpoint method

4.4. Computer vision performance

The results of extraction of the robot's position relative to the followed person by utilisation of computer vision techniques are presented in this section.

In order for the autonomous navigation to work properly, it needs to be ensured that the robot can steadily and correctly approximate its position in relation to the followed person. In order to test the performance of the CV system, special test was designed and executed. It consisted of creating a pre-determined path that a person would take in front of stationery robot. The person would be in the field of vision at all times, occasionally coming near its ends. The path consisted of:

- Forward/backward movement
- Sideways movement
- Combination of both, resulting in diagonal movement

The path would be enclosed within 3 by 3 meters square, placed 1 meter away from the robot. The diagram of the designed path can be seen in Figure 4-7. The path would be executed while the computer vision system is running, thus recording the person position in real time for purposes of later comparison.

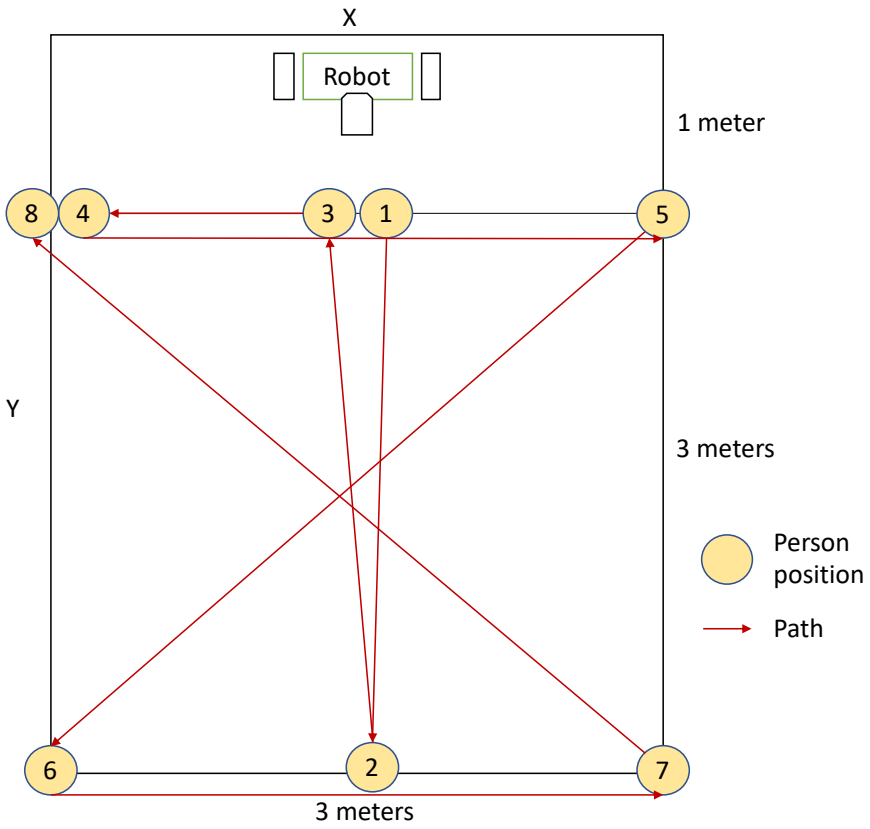

Figure 4-7 Pre-set path for computer vision position tracking evaluation

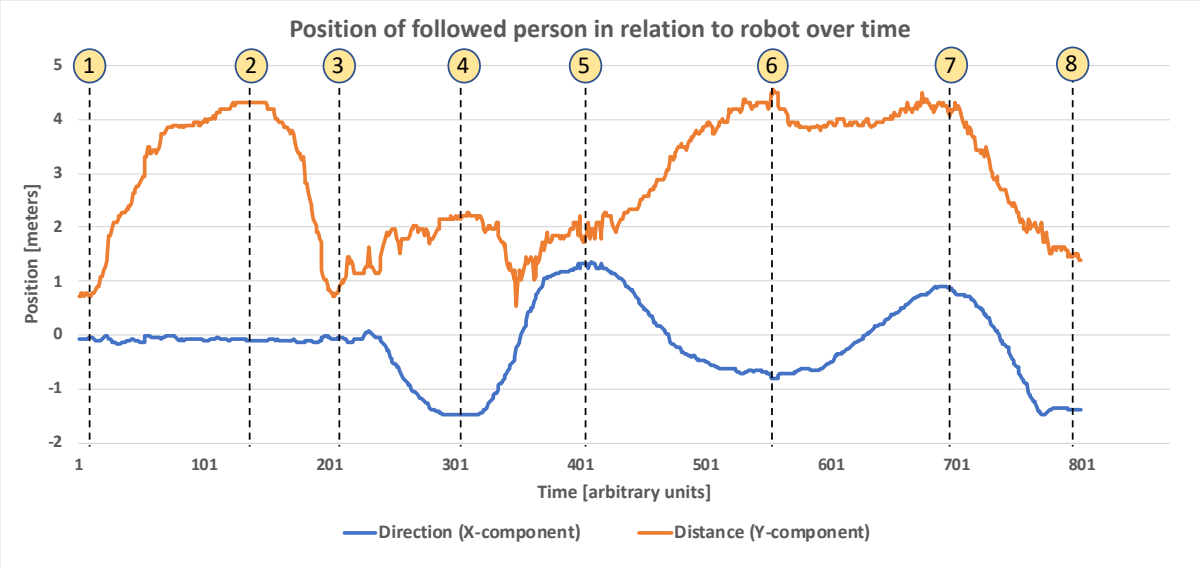

Figure 4-8 Tracking performance of the computer vision system

Figure 4-8 shows the data output of the computer vision system presenting the directional and distance position of the person in relation to the robot. According to Figure 4-7, the distance position is distance in Y direction and direction position is distance in X direction, together creating exact representation of person position in two-dimensional representation of the real world. The position values were converted from the CV outputs and normalized to the known path taken by the person, therefore the position can be extracted in meters, as opposed to pixel values outputted by CV system.

It can be seen that even though the computer vision system is using relatively simplistic position extraction procedure, outlined in Section 3.6.6, its outputs are relatively accurate and of high resolution. Stages of movement between consecutive nodes of the set path can be easily seen and overall the output isn't outstandingly noisy. It has to be noted that the measurements taken by the system are not changing in a linear fashion, as presented by the direction position max value reached in steps 4 and 5 and 7 and 8 respectively. In both of those pairs the followed person was at the maximum sideways displacement, however the system recorded the values of the latter pair to be only about 65% of the actual value. That discrepancy can be partially contributed to the fish-eye lens that is disturbing the image in a way that helps to capture more information in a frame, but also changes the linearity of the appearance change as the object is moving across the frame. The exact parameters of the computer vision system used in this test are presented in Table 4-1

Table 4-1 Computer vision system parameters

Object detector	Orig, frame resolution	Red. Frame resolution	Frames per second
MobileNet SSD	1080x720	400x225	~14 FPS

4.5. Autonomous person following

This section presents the performance of the autonomous navigation and person following capability.

Autonomous navigation of the robot is dependent on all robot systems, namely:

- Computer vision system providing positional data
- PID control system providing movement capability
- Behavioural algorithm determining the action robot would take in response to the input data

The data shown in this section is coming from a pre-set path presented in Figure 4-9 that was taken by the followed person. The path is as follows:

- Initially the robot is at position 0 and person is at position 1. The robot starts actuation process in order to be arrive within the vicinity of the followed person
- Once the robot arrives at stable position, person walks away in straight line to position 2
- Once the robot arrives at stable position, person takes a left turn and walks in straight line to position 3
- Once the robot arrives at stable position, person walks to position 4 and stays there

Therefore, the followed person proceeds to the next position only when the robot arrives within the person's vicinity and stabilises its position.

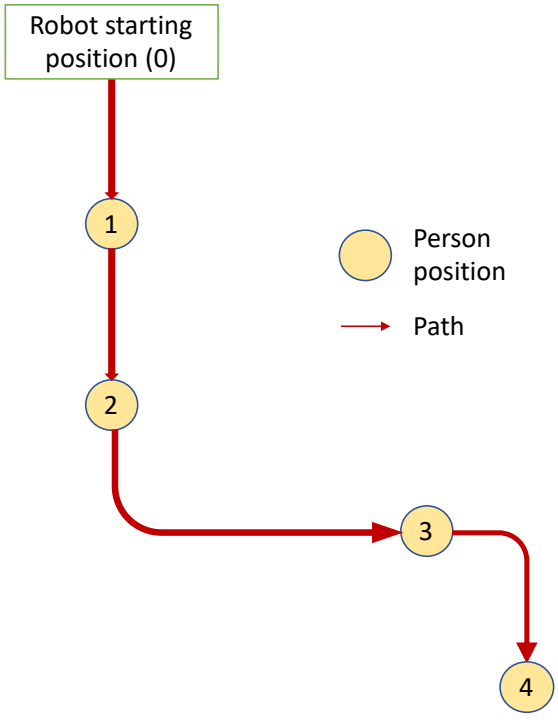

Figure 4-9 Autonomous behaviour test path

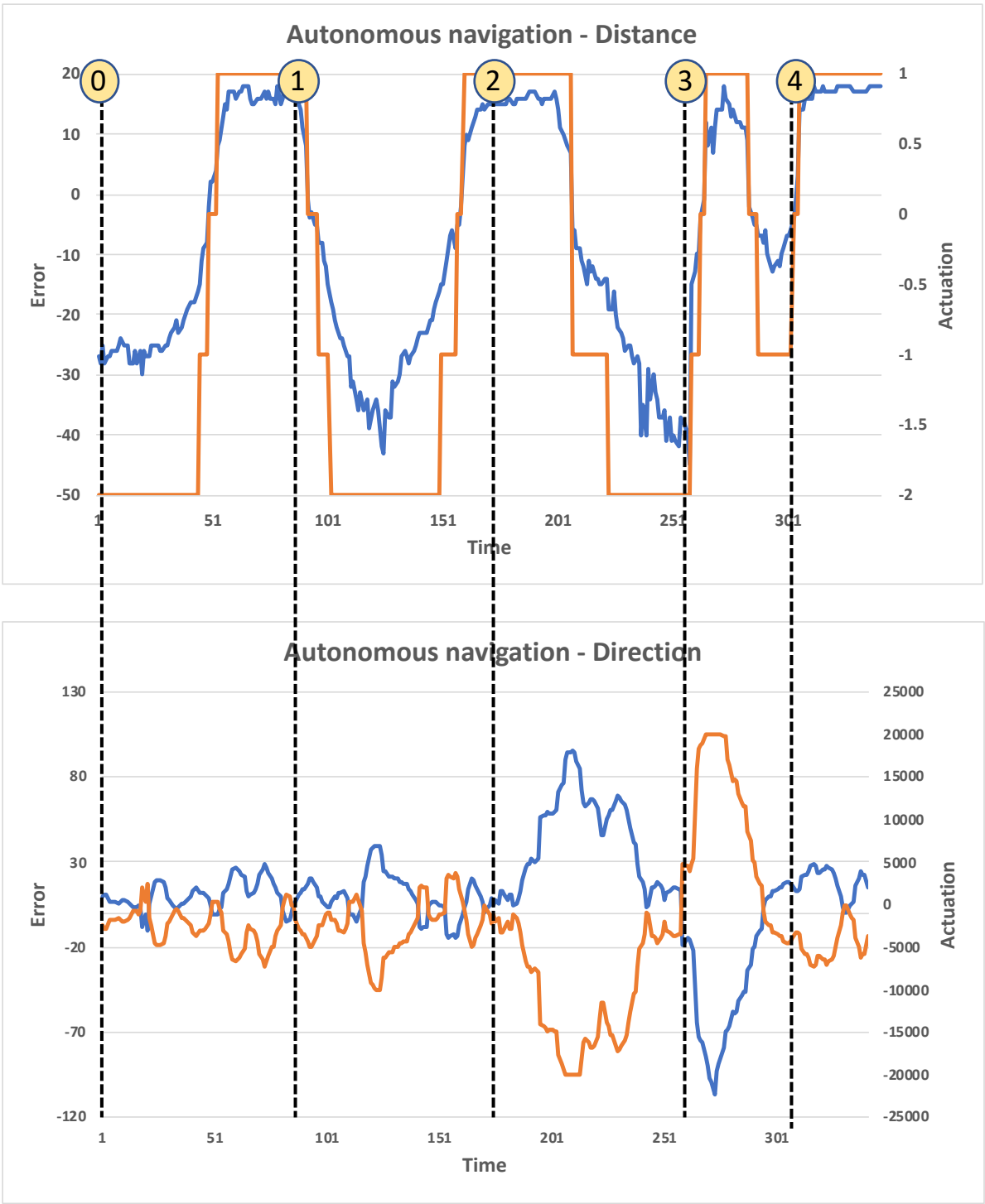

Figure 4-10 Robot's performance at autonomous navigation in person-following mode

Figure 4-10 presents the data gathered during the pre-set path test. The data is divided into controlled dimensions of distance and direction. It can be seen that in both cases the robot is actively trying to minimise the error, however with different approaches. In the distance correction, the robot can only work within the pre-set activation regions in order to prevent possible instabilities coming from implementation of more complicated controller. Additionally, the actuation was pre-set to be focused on correcting the negative error, which is representing the long relative distances. In reality, when the

robot is at position causing error of around 15, it is supposed to stay there, as the positive value of actuation was aimed at breaking, rather than moving away from the robot. If the robot stayed too close to the person for too long the movement of increasing the distance would be eventually executed with the 'breaking', or positive, value of the actuation.

Regarding the direction correction, it can be seen that the actuation is much more aggressive and directly proportional to the existing error. That results in much more dynamical correction of error and rather optimal performance.

4.6. Discussion

It was showed that the methodology outlined in Design methods can produce satisfactory outputs. The robot's build is very strong and durable and provides a sturdy and reliable platform for controller implementation and testing.

The balancing system based on the PID controller is sufficient in achieving the dynamic balance, however it has some noticeable drawbacks. Besides fine tuning the gain values, some additional components need to be added to the control system in order to achieve better performance, namely:

- Additional filtering of the IMU data. The sensor utilises the built-in complementary filter, which allows to achieve an accurate and stable reading. However, as the robot is of relatively big sizes, some flexing effects may be in place and cause external noise on the sensors data. Application of simple moving average for the reading should improving the miniature noise that may be causing the rapid oscillatory behaviour of the derivative action.
- Lead-lag compensator of the integral action. One of the biggest factors contributing towards the persistent oscillation in the system was the lag of the integral part relative to the pitch readings. In order to reduce the effect of that, the lag compensator should be added to increase the stability
- Low-pass filter of the rotary encoder data. The velocity PID proved to be inapplicable due to build-up of various imperfections in the system. One of the biggest factors for that was the noise signal from the rotary encoder, which in pair with derivative action caused spiralling

amplification closed-loop, causing instability. Low-pass filter or other comparable method should allow to implement the velocity PID

The computer vision system was working properly, achieving satisfactory performance at recognizing and tracking a person and extracting the positional data. Nevertheless, the position extraction could be further improved by addition of some form of distance sensors to the system. The inherit drawback of using the camera feed as data input is that a person or object of interest needs to always be in the frame and its whole body has to be within the field of view of camera, otherwise the bounding-box-based distance measurement is producing false readings. That could be improved if the readings were to be at least partially confirmed by ground truth distance data.

The autonomous navigation system has proven to be capable of executing the task of autonomously following a given person. However, its performance is far from ideal. There are several points of improvement:

- Implementing proper distance proportional controller instead of discrete actuation controller. The current system is capable of controlling the distance, however its behaviour is jerky, as the controller doesn't have a continuous spectrum of possible actuations to choose from, but rather a very limited choice of 3 possible actions that it can perform. Implementation of proportional distance control would smooth the robot response, however it may cause unexpected instabilities that were the main reason of going with discrete actuation option
- Expanding the decision tree. Currently the behaviour of the autonomous behaviour of the robot is very limited, with only a few possible outcomes. Expanding the decision tree to encompass more situations would provide safer operation of the robot and also get it closer to mimic a behaviour that a real living being could be characterised by.

5. Conclusions

This report outlined the work performed with an aim of creating a technological platform for personal robotic assistant. The workflow started with performing the literature review, that has helped to formulate the mathematical basis for mechanics of the two-wheeled, self-balancing (TWSB) robots and possible control options. The literature review also contained state-of-the-art from the field of TWSB robots and computer vision systems. That has helped to inform future design choices and guide the project into feasible directions. Next, the design methods were created detailing the path taken to conceive and execute the creation of each of the robot's subsystems. The methodology started with creating a top-level design of the robot, based on various inspirations and giving overall architecture and looks of the robot. Next, the rough sketches were transformed into detailed CAD model containing all the necessary parts. Based on the CAD model the robot was manufactured part by part. The drive and power trains were designed based on the actuation requirements, with particular components chosen to provide the robot with outstanding actuation performance. Next, the powertrain was connected with the brain of the robot in the form of microcontroller that was responsible for manipulating the sensory input information and compute appropriate control action based on the PID controller. In parallel, the design of computer vision system was developed. Particular libraries were found and parts of software connected together in order to produce people detection. The tracking system was created from scratch with twofold architecture of position and appearance tracking. Based on the positional outputs from the computer vision system the behavioural algorithm was conceived providing the robot with autonomous navigation capabilities. Consecutively the subsystems of the robot were tested and validated, starting with tuning of the PID controller, to evaluation of the self-balancing sequence and radio controlled drive. Next, the computer vision system was tested by capturing the position of tracked person during pre-set path. Building on that, the autonomous navigation was tested based on the pre-set path. All of the tests were passed successfully confirming proper workings of the robot.

Even though the robot isn't perfect in many ways, its performance can be taken as outstanding given the tight deadlines of the project and very intermediate level of the required knowledge that the owner of the project possessed prior to its start. Overall, the project is seemed as successful. As with everything, there is further work to be conducted, which is detailed in the next section.

5.1. Future recommendations

Recommendation regarding future work are divided into two groups. The first group contains the tasks outlined in the Discussion section, such as: further filtering of the IMU data, preliminary filtering of the rotary encoder, implementation of the lag compensator, addition of distance sensor, implementation of proportional distance control and further development of the behavioural decision tree.

The second group contains future developments activities. As mentioned in the Introduction section, the robot is supposed to serve as a technological platform for a personal assistant. In order to be efficient in manipulating the human environment, it requires some additional form of actuation. A concept was created that illustrated one of the possible applications of the robot, namely the personal home care. That concept involves creating robotic hands that would allow the robot to perform house chores or simply bring a glass of water to an immobile person. Additionally, in order to make the interaction with the robot more pleasurable, it is ought to be less machine-like. That would be achieved by creating an outer shell that would protect the electronical insides of the robot. Moreover, an interactive screen would be added through which a non-verbal communication may be performed. The concept is visualised in Figure 5-1

Figure 5-1 Future vision of the robot - Puter, the personal home care robot

6. Appendix

6.1. Appendix 3 - Microcontroller code (C)

```
#include <Wire.h>  //Include the Wire.h library so we can communicate with the
gyro
#include <PID_v1.h>

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>


#define THROTTLE_SIGNAL_IN 9 // INTERRUPT 0 = DIGITAL PIN 2 - use the
interrupt number in attachInterrupt
#define THROTTLE_SIGNAL_IN_PIN 9 // INTERRUPT 0 = DIGITAL PIN 2 - use the PIN
number in digitalRead

#define NEUTRAL_THROTTLE 1500 // this is the duration in microseconds of
neutral throttle on an electric RC Car

#define DIR_SIGNAL_IN 10 // INTERRUPT 0 = DIGITAL PIN 2 - use the interrupt
number in attachInterrupt
#define DIR_SIGNAL_IN_PIN 10 // INTERRUPT 0 = DIGITAL PIN 2 - use the PIN
number in digitalRead

#define NEUTRAL_DIR 1500 // this is the duration in microseconds of neutral
throttle on an electric RC Car
volatile int nThrottleIn = NEUTRAL_THROTTLE; // volatile, we set this in the
Interrupt and read it in loop so it must be declared volatile
volatile unsigned long ulStartPeriod = 0; // set in the interrupt
volatile boolean bNewThrottleSignal = false; // set in the interrupt and read
in the loop

volatile int nDirIn = NEUTRAL_DIR; // volatile, we set this in the Interrupt
and read it in loop so it must be declared volatile
volatile unsigned long ulStartPeriodDir = 0; // set in the interrupt
volatile boolean bNewDirSignal = false; // set in the interrupt and read in
the loop

#include <ODriveArduino.h>
float pid_signal;
// Printing with stream operator
template<class T> inline Print& operator <<(Print &obj,     T arg) {
  obj.print(arg);
  return obj;
}
template<>        inline Print& operator <<(Print &obj, float arg) {
  obj.print(arg, 4);
  return obj;
}

bool dir_bool = true;
bool dist_bool = true;

int horP = 250;
int horI = 0;
```

```
int horD = 0;
int vel = 0;
//String data;
int dirINT;
int distINT;
float CV_vel_setpoint = 0;
float CV_setpoint = 0;
float CV_vel = 0;

boolean distFLAG_FOR = 0;
boolean distFLAG_BACK = 0;

double Setpoint, Input, Output;
PID horPID(&Input, &Output, &Setpoint, horP, horI, horD, DIRECT);
// ODrive object
ODriveArduino odrive(Serial1);

long vel0;
long vel1;
long ave_vel = 0;
long vel0past = 0;
float TF = 0.1;
int pos0;
int pos1;
float curr0;
float curr1;

int gyro_address = 0x68;                                        //MPU-6050 I2C
address (0x68 or 0x69)
float yaw;
float pitch;
float roll;

RF24 radio(9, 10) ; // CE, CSN
const byte address[6] = "00001";

int pid_setpoint = 0;
int vel_setpoint = 0;
float kp = 0.5; //10000                                         //Gain setting
for the P-controller (15)
float kd = 120; //1500  100                                      //Gain setting
for the I-controller (1.5)
float ki = 0.03; //0.01 //100                                        //Gain
setting for the D-controller (30
float vel_kp = 0.08;
float vel_kd = 0;
float vel_ki = 0; //0.001
long counter = 0;
float pid_adjustment = 0;
float lastError;
float cumError = 0;
float cumErrorUse = 0;
float previousTimePID;

float vel_lastError;
float vel_cumError = 0;
float vel_cumErrorUse = 0;

float vel_pitch = 0;
```

```
int rec_CV;
long rc_values[2];
int knob1;
int knob2;
int butt;
int buttState = 0;
int pot;

float RC_setpoint = 0;
float RC_vel_setpoint = 0;

float left_motor_signal = 0;
float right_motor_signal = 0;



//motor 0 is left, motor 1 is right


// ================================================================
// ===               MPU6050                ===
// ================================================================

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h
files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation
board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
```

```
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
//#define OUTPUT_TEAPOT



#define INTERRUPT_PIN 14  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 =
success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor
measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor
measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and
gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0x00, 0x00,
'\r', '\n' };



// ================================================================
// ===               MPU INTERRUPT DETECTION ROUTINE                ===
// ================================================================
```

```
volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin
has gone high
void dmpDataReady() {
  mpuInterrupt = true;
}



// ================================================================
// ===                      MPU INITIAL SETUP                     ===
// ================================================================

void MPUInitialize() {
  // join I2C bus (I2Cdev library doesn't do this automatically)
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  Wire.begin();
  Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having
compilation difficulties
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif

  // initialize serial communication
  // (115200 chosen because it is required for Teapot Demo output, but it's
  // really up to you depending on your project)
  Serial.begin(115200);
  //while (!Serial); // wait for Leonardo enumeration, others continue
immediately

  // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or Arduino
  // Pro Mini running at 3.3V, cannot handle this baud rate reliably due to
  // the baud timing being too misaligned with processor ticks. You must use
  // 38400 or slower in these cases, or use some kind of external separate
  // crystal solution for the UART timer.

  // initialize device
  Serial.println(F("Initializing I2C devices..."));
  mpu.initialize();
  pinMode(INTERRUPT_PIN, INPUT);

  // verify connection
  Serial.println(F("Testing device connections..."));
  Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") :
F("MPU6050 connection failed"));


  // load and configure the DMP
  Serial.println(F("Initializing DMP..."));
  devStatus = mpu.dmpInitialize();

  // supply your own gyro offsets here, scaled for min sensitivity
  mpu.setXGyroOffset(-3491);
  mpu.setYGyroOffset(-20);
  mpu.setZGyroOffset(-41);
  mpu.setZAccelOffset(1014); // 1688 factory default for my test chip

  // make sure it worked (returns 0 if so)
  if (devStatus == 0) {
```

```
    // Calibration Time: generate offsets and calibrate our MPU6050
    mpu.CalibrateAccel(6);
    mpu.CalibrateGyro(6);
    mpu.PrintActiveOffsets();
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection
    Serial.print(F("Enabling interrupt detection (Arduino external interrupt
"));
    Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
    Serial.println(F(")..."));
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady,
RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's okay to
use it
    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady = true;

    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
  }

  // configure LED for output
  pinMode(LED_PIN, OUTPUT);
}



// ================================================================
// ===                      MPU MAIN LOOP                       ===
// ================================================================

void readAngles() {
  // if programming failed, don't try to do anything
  if (!dmpReady) return;
  // read a packet from FIFO
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet

#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    yaw = ypr[0] * 180 / M_PI;
    pitch = ypr[1] * 180 / M_PI;
    roll = ypr[2] * 180 / M_PI;
    /*good
```

```
    Serial.print("ypr ");
    Serial.print(yaw);
    Serial.print(" ");
    Serial.print(pitch);
    Serial.print(" ");
    Serial.print(roll);*/
#endif

    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
  }
}

// ===============================================================
// ===                ODRIVE                ===
// ===============================================================


void OdriveInitiation() {
  // ODrive uses 115200 baud
  Serial1.setRX(27);
  Serial1.setTX(1);
  Serial1.begin(115200);


  //while (!Serial) ; // wait for Arduino Serial Monitor to open

  //Serial.println("ODriveArduino");
  //Serial.println("Setting parameters...");

  // In this example we set the same parameters to both motors.
  // You can of course set them different if you want.
  // See the documentation or play around in odrivetool to see the available
parameters
  for (int axis = 0; axis < 2; ++axis) {
    Serial1 << "w axis" << axis << ".controller.config.vel_limit " <<
500000.0f << '\n';
    Serial1 << "w axis" << axis << ".motor.config.current_lim " << 50.0f <<
'\n';
    Serial1 << "w axis" << axis << ".motor.config.calibration_current " <<
20.0f << '\n';
    //odrive_serial << "w axis" << axis <<".motor.config.pre_calibrated =
True"<<'\n'; //tells odrive its motor is calibrated
    //odrive_serial << "w axis" << axis <<".encoder.config.pre_calibrated =
True"<<'\n';
    // This ends up writing something like "w axis0.motor.config.current_lim
10.0\n"
  }

  //ODRIVE calibration
  for (int c = 0; c < 2; c++) {
    int motornum = c;
    int requested_state;

    requested_state = ODriveArduino::AXIS_STATE_MOTOR_CALIBRATION;
    Serial << "Axis" << c << ": Requesting state " << requested_state << '\n';
    odrive.run_state(motornum, requested_state, false);
    delay(10000);
```

```
    requested_state = ODriveArduino::AXIS_STATE_ENCODER_OFFSET_CALIBRATION;
    Serial << "Axis" << c << ": Requesting state " << requested_state << '\n';
    odrive.run_state(motornum, requested_state, false);
    delay(10000);
    /*requested_state = ODriveArduino::AXIS_STATE_FULL_CALIBRATION_SEQUENCE;
    Serial << "Axis" << c << ": Requesting state " << requested_state << '\n';
    odrive.run_state(motornum, requested_state, false);
    delay(10000); */
    requested_state = ODriveArduino::AXIS_STATE_CLOSED_LOOP_CONTROL;
    Serial << "Axis" << c << ": Requesting state " << requested_state << '\n';
    odrive.run_state(motornum, requested_state, false); // don't wait
    Serial.print("Motor: ");
    Serial.print(c);
    Serial.println(" calibrated");
  }
}

void OdriveWrite() {
  RC_vel_setpoint /= 10000;
  CV_vel_setpoint = CV_vel/10000;
  float left_motor_curr = (pid_signal - RC_vel_setpoint - CV_vel_setpoint);
//+ RC_vel_setpoint //map(left_motor_signal, -1000, 10000, -22000, 22000);
//we can either use mapping here or tweak with PID
  float right_motor_curr = -(pid_signal + RC_vel_setpoint + CV_vel_setpoint);
//map(right_motor_signal, -10000, 10000, -22000, 22000); //gain values to
amplify the PID output signal so
  if (abs(left_motor_curr) > 0.2) {
    odrive.SetCurrent(0, left_motor_curr);
    odrive.SetCurrent(1, right_motor_curr);
// it is within the velocity range of ODRIVE
  }

  vel0 = odrive.GetVelocity(0);
  vel1 = odrive.GetVelocity(1);
  ave_vel = abs(vel0 - vel1)/2;
  //Serial.print("ave_vel: ");
  //Serial.println(ave_vel);
  //Serial.print("vel");
  //Serial.print(ave_vel);

  /*good
  Serial.print(" Curr left: ");
  Serial.print(left_motor_curr);
  Serial.print(" Curr right: ");
  Serial.print(-right_motor_curr);
  Serial.print(" Vel left: ");
  Serial.print(vel0);
  Serial.print(" Vel right: ");
  Serial.println(-vel1); */


  /*Serial.print(" curr0: ");
  Serial.print(curr0);
  Serial.print(" curr1: ");
  Serial.println(curr1);*/
}


// ================================================================
```

```
// ===               RADIO COMMUNICATION               ===
// ================================================================

void calcInput()
{
  // if the pin is high, its the start of an interrupt
  if(digitalRead(THROTTLE_SIGNAL_IN_PIN) == HIGH)
  {
    // get the time using micros - when our code gets really busy this will
become inaccurate, but for the current application its
    // easy to understand and works very well
    ulStartPeriod = micros();
  }
  else
  {
    // if the pin is low, its the falling edge of the pulse so now we can
calculate the pulse duration by subtracting the
    // start time ulStartPeriod from the current time returned by micros()
    if(ulStartPeriod && (bNewThrottleSignal == false))
    {
      nThrottleIn = (int)(micros() - ulStartPeriod);
      ulStartPeriod = 0;

      // tell loop we have a new signal on the throttle channel
      // we will not update nThrottleIn until loop sets
      // bNewThrottleSignal back to false
      bNewThrottleSignal = true;
    }
  }
}

void calcInputDir()
{
  // if the pin is high, its the start of an interrupt
  if(digitalRead(DIR_SIGNAL_IN_PIN) == HIGH)
  {
    // get the time using micros - when our code gets really busy this will
become inaccurate, but for the current application its
    // easy to understand and works very well
    ulStartPeriodDir = micros();
  }
  else
  {
    // if the pin is low, its the falling edge of the pulse so now we can
calculate the pulse duration by subtracting the
    // start time ulStartPeriod from the current time returned by micros()
    if(ulStartPeriodDir && (bNewDirSignal == false))
    {
      nDirIn = (int)(micros() - ulStartPeriodDir);
      ulStartPeriodDir = 0;

      // tell loop we have a new signal on the throttle channel
      // we will not update nThrottleIn until loop sets
      // bNewThrottleSignal back to false
      bNewDirSignal = true;
    }
  }
}
```

```
void radioInitialize() {
  attachInterrupt(THROTTLE_SIGNAL_IN,calcInput,CHANGE);
  attachInterrupt(DIR_SIGNAL_IN,calcInputDir,CHANGE);
  pinMode(A9, INPUT);
  Serial.begin(115200);
}

void radioReceive() {
  pot = analogRead(A9);

   // if a new throttle signal has been measured, lets print the value to
serial, if not our code could carry on with some other processing
 if(bNewThrottleSignal)
 {

   //Serial.println((String)" Throttle: " + nThrottleIn);


   // set this back to false when we have finished
   // with nThrottleIn, while true, calcInput will not update
   // nThrottleIn
   bNewThrottleSignal = false;
 }
 if(bNewDirSignal)
 {

   //Serial.println((String) " Dir: " + nDirIn);


   // set this back to false when we have finished
   // with nThrottleIn, while true, calcInput will not update
   // nThrottleIn
   bNewDirSignal = false;
 }

    if (nDirIn > 1510 || nDirIn < 1490) {
      RC_vel_setpoint = map(nDirIn, 1000, 1900, -30000, 30000);
    }
    else {
      RC_vel_setpoint = 0;
    }


    if (nThrottleIn > 1520 || nThrottleIn < 1480) {
      RC_setpoint = map(nThrottleIn, 1000, 1900, -40000, 40000);
    }
    else {
      RC_setpoint = 0;
    }

  if(pot > 990){
    buttState = 0;
  }
  else{
    buttState = 1;
  }


  //Serial.print((String)"Knob1: " + RC_setpoint + " , ");
```

```
  //Serial.print((String)"Knob2: " + RC_vel_setpoint + " , ");

}

// ================================================================
// ===               PID                ===
// ================================================================
String getValue(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

void AI_PIDsetup(){
  Setpoint = 0;
  horPID.SetMode(AUTOMATIC);
  horPID.SetOutputLimits(-20000, 20000);
  Serial.setTimeout(5);
}
void AI_PIDupdate(){
  if (Serial.available() > 0) {
    String data = Serial.readString();
    String dist = getValue(data, ':', 0);
    String dir = getValue(data, ':', 1);
    //String dir = Serial.readStringUntil("/n");
    dirINT = dir.toInt();
    distINT = dist.toInt();
    //distINT = 90;
    Serial.print("DIR:");
    Serial.print(dirINT);
    Serial.print(",");
    Serial.print("DIST:");
    Serial.print(distINT);
    Serial.print(",");


    if (dir_bool == true){
    Input = dirINT;
    horPID.Compute();
    CV_vel = Output;
    Serial.print("VEL:");
    Serial.print(CV_vel);
    Serial.print(",");

    if (ave_vel != 0){

      if (ave_vel < 20000){
      CV_vel = CV_vel;
      }
```

```
  if (ave_vel > 20000 && ave_vel < 50000){
  CV_vel = CV_vel / 2;
  }
  if (ave_vel >50000){
  CV_vel = CV_vel / 4;
  }


}


//Serial.print("You sent me: ");
//Serial.println(CV_vel);
}
if (distINT <= 50 ){
  distFLAG_FOR = 1;
}
else if (distINT >= 80){
  distFLAG_FOR = 0;

}

if (distINT >= 95 ){
  distFLAG_BACK = 1;
}
else if (distINT <= 80){
  distFLAG_BACK = 0;

}


if (dist_bool == true){
  if (distFLAG_FOR = 1 && distINT < 65){
    CV_setpoint = -2;
    //Serial.print("CV_set: ");
    //Serial.println(CV_setpoint);
    Serial.print("CV");
    Serial.print(CV_setpoint);
    Serial.print(",");
  }
  else if(distFLAG_FOR = 1 && distINT >= 65 && distINT < 75){
    CV_setpoint = -1;
    //Serial.print("CV_set: ");
    //Serial.println(CV_setpoint);
    Serial.print("CV");
    Serial.print(CV_setpoint);
    Serial.print(",");
  }

  else if(distFLAG_FOR = 1 && distINT >= 75 && distINT < 80){
    CV_setpoint = 0;
    //Serial.print("CV_set: ");
    //Serial.println(CV_setpoint);
    Serial.print("CV");
    Serial.print(CV_setpoint);
    Serial.print(",");
  }
```

```
      if (distFLAG_BACK = 1 && distINT > 85){
        CV_setpoint = 1;
        //Serial.print("CV_set: ");
        //Serial.println(CV_setpoint);
        Serial.print("CV");
        Serial.print(CV_setpoint);
        Serial.print(",");
      }
      else if (distFLAG_BACK = 1 && distINT > 80 && distINT <= 85){
        CV_setpoint = 0;
        //Serial.print("CV_set: ");
        //Serial.println(CV_setpoint);
        Serial.print("CV");
        Serial.print(CV_setpoint);
        Serial.print(",");
      }

    }
    Serial.println("END");
  }

}
float computePID() {
  int currentTimePID = millis();
  if (counter = 0) {
    previousTimePID = currentTimePID;
  }
  int elapsedTime = currentTimePID - previousTimePID;
  counter += 1;
  /*
  vel0 = odrive.GetVelocity(0);
  vel0 /= 1000;
  vel0 = vel0 * TF + (1-TF) * vel0past;
  RC_setpoint /= 1000;
  float vel_error = vel_setpoint + RC_setpoint - vel0;
  vel_cumError += vel_error * elapsedTime / 10;

  if (vel_cumError > 1000) {
    vel_cumErrorUse = 1000;
  }
  else if (vel_cumError < -1000) {
    vel_cumErrorUse = -1000;
  }
  else {
    vel_cumErrorUse = cumError;
  }

  float vel_rateError = (vel_error - vel_lastError) / elapsedTime;
  float vel_pid_output = vel_kp * vel_error + vel_ki * vel_cumErrorUse +
vel_kd * vel_rateError; //that needs to be adjusted to degrees
  */
  RC_setpoint /= 10000;
  float error =  pid_setpoint + RC_setpoint + CV_setpoint - pitch ; // -
vel_pid_output + vel_pid_output + pid_adjustment + vel_pid_output +
pid_input_MC
  if( buttState == 1){
    cumError += error * elapsedTime /10 ;
  }
```

```
  else{
    cumError = 0;
  }

  if (cumError > 1000) {
    cumErrorUse = 1000;
  }
  else if (cumError < -1000) {
    cumErrorUse = -1000;
  }
  else {
    cumErrorUse = cumError;
  }

  float rateError = (error - lastError) / elapsedTime;

  float pid_output = kp * error + ki * cumErrorUse + kd * rateError;
  /*
  Serial.print(" cumError: ");
  Serial.print(cumError);
  Serial.print(" , ");
  Serial.print(" kiAct: ");
  Serial.print(ki * cumErrorUse);
  Serial.print(" , ");
  Serial.print(" kpAct: ");
  Serial.print(kp * error);
  Serial.print(" , ");
  Serial.print(" pidAdj: ");
  Serial.print(pid_adjustment);
  */
  /*Serial.print(" velError: ");
  Serial.print(vel_error);
  Serial.print(" velPID: ");
  Serial.print(vel_pid_output);*/

  /* GOOD
  Serial.print(" cumError: ");
  Serial.print(cumError);
  Serial.print(" Ki: ");
  Serial.print(cumErrorUse * ki);

  Serial.print(" kpAct: ");
  Serial.print(kp * error);
  Serial.print(" KD: ");
  Serial.print(kd * rateError);
  Serial.print(" PID: ");
  Serial.print(pid_output);
  Serial.print(" T: ");
  Serial.print(elapsedTime); */
  previousTimePID = currentTimePID;
  lastError = error;
  vel0past = vel0;
  //vel_lastError = vel_error;
  return pid_output;


}
void computeWheelVelocity(int pid_output) {
  //int total_input = side_input_MC; //+ side_input_CV;
```

```
  //left_motor_signal = pid_output + total_input; //map MC and CV values
respectively so that the pos on the left is negative and on the right positive
  //right_motor_signal = pid_output - total_input;
}

// ================================================================
// ===               ACTUAL PROGRAM                ===
// ================================================================


void setup() {
  Serial.begin(115200);
//Start the serial port at 9600 kbps
  Wire.begin();
//Start the I2C bus as master
  MPUInitialize();
  radioInitialize();
  OdriveInitiation(); // initialize and calibrate
  AI_PIDsetup();
  readAngles();
}


void loop() {

  radioReceive();
  readAngles();
  pid_signal = computePID();
  AI_PIDupdate();
  //computeWheelVelocity(pid_signal);
  OdriveWrite();
  //OdriveRead();

}
```

7. References

Hellman, H., Sunnerman, H., (2015) “Two-Wheeled Self-Balancing Robot” *KTH Royal Institute of Technology in Stockholm [online]* available at “https://kth.diva-portal.org/smash/get/diva2:916184/FULLTEXT01.pdf”

Alyamkin, S., Ardi, M., Berg, A., Brighton, A., Chen, B., Chen, Y., Cheng, H., Fan, Z., Feng, C., Fu, B., Gauen, K., Goel, A., Goncharenko, A., Guo, X., Ha, S., Howard, A., Hu, X., Huang, Y., Kang, D., Kim, J., Ko, J., Kondratyev, A., Lee, J., Lee, S., Lee, S., Li, Z., Liang, Z., Liu, J., Liu, X., Lu, Y., Lu, Y., Malik, D., Nguyen, H., Park, E., Repin, D., Shen, L., Sheng, T., Sun, F., Svitov, D., Thiruvathukal, G., Zhang, B., Zhang, J., Zhang, X. and Zhuo, S. (2020) *Low-Power Computer Vision: Status, Challenges, Opportunities* [online] available from <https://arxiv.org/abs/1904.07714> [13 February 2020]

Bengio, Y. (2009) "Learning Deep Architectures For AI". *Foundations And Trends® In Machine Learning* 2 (1), 1-127

COCO (2020) *COCO - Common Objects In Context* [online] available from <http://cocodataset.org/#home> [1 February 2020]

Juang, H. and Lum, K. (2013) "Design And Control Of A Two-Wheel Self-Balancing Robot Using The Arduino Microcontroller Board". *2013 10Th IEEE International Conference On Control And Automation (ICCA)*

Kyrykovych, A. (2020) *Deep Neural Networks - Kdnuggets* [online] available from <https://www.kdnuggets.com/2020/02/deep-neural-networks.html> [1 March 2020]

Pao, C. (2018) *The Importance Of IMU Motion Sensors - CEVA’S Experts Blog* [online] available from <https://www.ceva-dsp.com/ourblog/what-is-an-imu-sensor/> [1 March 2020]

Rahman, M., Ashik-E-Rasul, Haq, N., Hassan, M., Hasib, I. and Hassan, K. (2016) *Development Of A Two Wheeled Self Balancing Robot With Speech Recognition And Navigation Algorithm*.

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. (2020) *You Only Look Once: Unified, Real-Time Object Detection* [online] available from <https://arxiv.org/abs/1506.02640> [29 February 2020]

Sharma, K. and Thakur, N. (2017) "A Review And An Approach For Object Detection In Images". *International Journal Of Computational Vision And Robotics* 7 (1/2), 196

Hassan, M., Khan, A., Khan, M., Uzair, M. and Khurshid, K. (2016) "A Computationally Low Cost Vision Based Tracking Algorithm For Human Following Robot". *2016 2Nd International Conference On Control, Automation And Robotics (ICCAR)*

Islam, M., Hong, J. and Sattar, J. (2019) "Person-Following By Autonomous Robots: A Categorical Overview". *The International Journal Of Robotics Research* 38 (14), 1581-1618

Juang, H. and Lurrr, K. (2013) "Design And Control Of ISBN: 978-953-7619-21-3 A Two-Wheel Self-Balancing Robot Using The Arduino Microcontroller Board". *2013 10Th IEEE International Conference On Control And Automation (ICCA)*

Park, J. and Cho, B. (2018) "Development Of A Self-Balancing Robot With A Control Moment Gyroscope". *International Journal Of Advanced Robotic Systems* 15 (2), 172988141877086

Pratama, D., Binugroho, E. and Ardilla, F. (2015) "Movement Control Of Two Wheels Balancing Robot Using Cascaded PID Controller". *2015 International Electronics Symposium (IES)*

Rahman, M., Ashik-E-Rasul, Haq, N., Hassan, M., Hasib, I. and Hassan, K. (2016) *Development Of A Two Wheeled Self Balancing Robot With Speech Recognition And Navigation Algorithm*.

Person-Following Robot". *2016 35Th Chinese Control Conference (CCC)*

Varghese, E., Vincent, A. and Bagyaveereswaran, V. (2017) "Optimal Control Of Inverted Pendulum System Using PID Controller, LQR And MPC". *IOP Conference Series: Materials Science And Engineering* 263, 052007

Zimit, A., Yap, H., Hamza, M., Siradjuddin, I., Hendrik, B. and Herawan, T. (2018) "Modelling And Experimental Analysis Two-Wheeled Self Balance Robot Using PID Controller". *Computational Science And Its Applications – ICCSA 2018* 683-698

Ogata, K. (2016) *Modern Control Engineering*. [Delhi]: Pearson

Zhihui, X. (2008) *Computer Vision*. SL: Sciyo.com

Ghani, Nor Maniha Abdul et al. “Two Wheels Balancing Robot with Line Following Capability.” (2011).

Takafumi Sonoura, Takashi Yoshimi, Manabu Nishiyama, Hideichi Nakamoto, Seiji Tokura and Nobuto Matsuhira (2008*). Person Following Robot with Vision-based and Sensor Fusion Tracking Algorithm*, Computer Vision, Xiong Zhihui (Ed.), ISBN: 978-953-7619-21-3, InTech,

Siradjuddin, Indrazno & Setiawan, Budhy & Fahmi, Ahmad & Amalia, Zakiyah & Erfan, ROHADI. (2017). State space control using LQR method for a cart-inverted pendulum linearised model. 17. 119-126.
Bageant, Maia. (2012). Balancing a Two-Wheeled Segway Robot.
Razmjooy, Navid & A., Madadi & Alikhani, Hamid & Mohseni, Mona. (2014). Comparison of LQR and Pole Placement Design Controllers for Controlling the Inverted Pendulum. journal of world electrical engineering and technology. 3.

Peters, James & Pal, Sankar. (2010). Cantor, fuzzy, near, and rough sets in image analysis. Rough Fuzzy Image Analysis: Foundations and Methodologies.
Color Model (2020) available from <https://en.wikipedia.org/wiki/Color_model> [29 April 2020]

HSL And HSV (2020) available from <https://en.wikipedia.org/wiki/HSL_and_HSV> [29 March 2020]

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. and Berg, A. (2016) "SSD: Single Shot Multibox Detector". *Computer Vision – ECCV 2016* 21-37

PacificIntegrated (2020) *Bosch Aluminum Structural Framing System | Pacific Integrated Handling* [online] available from <http://www.pacificintegrated.com/catalog/bosch-aluminum-structural-framing-system> [29 April 2020]

Schmoeller da Roza, F., Ghizoni da Silva, V., Pereira, P. and Wildgrube Bertol, D. (2016) "Modular Robot Used As A Beach Cleaner". *Ingeniare. Revista Chilena De Ingeniería* 24 (4), 643-653

Wikipedia (2020) *K-Means Clustering* [online] available from <https://en.wikipedia.org/wiki/K-means_clustering> [29 April 2020]

Wikipedia (2020) *RGB Color Model* [online] available from <https://en.wikipedia.org/wiki/RGB_color_model> [29 April 2020]

Zielinska, T., Gao, Z., Zurawska, M., Zheng, Q., Mergner, T. and Lippi, V. (2017) "Postural Balance Using A Disturbance Rejection Method". *2017 11Th International Workshop On Robot Motion And Control (Romoco)*