Heriot-Watt University
School of Engineering and Physical Sciences

Modelling, Simulation and control of a humanoid robot

Inserting a pin to a hole using DARWINOP

Group members: Roshenac Mitchell, Markus Lampert, Nurgaliyev Shakh-Izat, Samir Mammadov, Hugo Sardinha, Waqar Khadas

Table of Contents

1 Introduction

A **humanoid robot** is a robot that has a general structure of the human body, such as two legs, two arms, a torso, and a head (Williams, 2004). Although, some shapes of a humanoid robot may not be exactly the same as that of a human, a humanoid robot has a basic similar appearance and functions of a human. For its human-like features described above, a humanoid robot has a potential to conduct tasks in human environments. Furthermore, a humanoid robot may even use tools designed for a human without modification. The development purpose of a humanoid robot is to make a robot that thinks and acts like a human (Williams, 2004). At the end, the humanoid robot will do work on behalf of a human, and a human can concentrate on more productive activities. The other significant development purpose is to understand mental and physical fundamentals of a human. So, many researchers from different fields adopt humanoid robots for their research platform for its synthetic characteristics.

The aim of this project is inserting a pin to a hole using Darwin-Op robot. To solve this iterative problem, a humanoid must be constructed considering expandable-modifiable system structure, high performance, simple maintenance, familiar development environment, and affordable prices. Therefore, in this paper, we suggest the design method for humanoid platform DARwin-OP (Dynamic Anthropomorphic Robot with Intelligence - Open Platform) as shown in Fig. 1 which has a network based modular structure and a standard PC architecture to meet above requirements (Ha, Tamura and Asama, 2013).

Darwin-OP is an affordable, miniature-humanoid-robot platform with advanced computational power, sophisticated sensors, high payload capacity, and dynamic motion ability to enable many exciting research and education activities (Ha, Tamura and Asama, 2013).

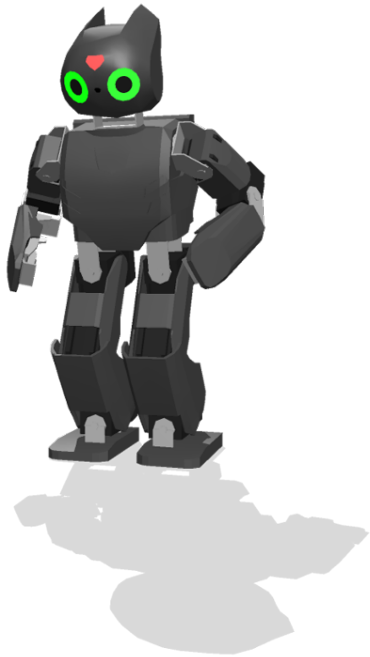


Figure 1: Darwin Robot

1.1 System overview

DARwIn-OP has a network-based modular structure and a standard PC architecture, as shown in Figure 2. All devices, such as actuators, sensors, LEDs, buttons, and external I/Os, are connected to the sub-controller by a serial bus network which fully supports DYNAMIXEL protocol (Ha, Tamura and Asama, 2013). Each device has a memory-mapped operation structure with designated ID. For the main controller, we adopted the Intel's ATOM Z530 CPU, normally used for netbooks (Ha, Tamura and Asama, 2013). The main controller communicates with the sub-controller by USB. The sub-controller works as a gateway to access devices. Therefore, all

devices are encapsulated as an USB device, which means that the development environment is just like a standard PC (Ha, Tamura and Asama, 2013).

1.2 Specification

- Default walking speed: 24.0 cm/sec (9.44 in/sec) 0.25 sec/step - user modifiable gait
- Default standing up time from ground: 2.8 sec (from facing down) and 3.9 sec (from facing up)
- user modifiable speed
- Built-in PC: 1.6 GHz Intel Atom Z530 on-board 4GB flash SSD
- Management controller (CM-730): ARM CortexM3 STM32F103RE 72MHz
- 20 actuator modules (6 DOF leg x2+ 3 DOF arm x2 + 2 DOF neck)
- Actuators with durable metallic gears (DYNAMIXEL MX-28)
- Self-maintenance kit (easy to follow steps and instructions)
- Standby mode for low power consumption
- 3Mbps high-speed Dynamixel bus for joint control
- Battery (30 minutes of operations), charger, and external power adapter
  (Battery can be removed from robot without shutting down by plugging in external power before removal)
- Versatile functionality (can accept legacy, current, and future peripherals)
- 3-axis gyro, 3-axis accelerometer, button x3, detection microphone x2

Figure 2: Darwin-op structure

2 Robot design and simulation

We will perform the modeling, simulation and control of a humanoid robot – DARWIN to give some demonstrations to show the capability of the humanoid robot. It will cover the simulation of the motion of the robot using Webot, Solidworks , Proteus and C++ software. This model must include at least the servo module, trajectory planning module, inverse kinematics module and all forward kinematic module if needed.

2.1 The Robotics Simulators

A robotics simulator is used to create embedded applications for a robot without depending on the actual physical machine, thus saving cost and time. These applications can be transferred onto the real robot without modifications. The use of a robotics simulator for development of a robotics control program is highly recommended regardless of whether an actual robot is available or not. The simulator allows for robotics programs to be conveniently written and debugged off-line with the final version of the program tested on an actual robot. Some examples of robot Simulators are listed below:

- Webot
- V-REP
- Gazebo etc.

2.1.1 Webot Robot Simulator

Webots is a professional robot simulator widely used for educational purposes. It uses the ODE (Open Dynamics Engine) for detecting of collisions and simulating rigid body dynamics. The ODE library allows one to accurately simulate physical properties of objects such as velocity, inertia and friction. In addition, it is also possible to build new models from scratch. When designing a robot model, we specify both the graphical and the physical properties of the objects. The graphical properties:

- shape
- dimensions
- position
- orientation
- colors
- texture of the object

The physical properties:

- mass,
- friction factor as well as the spring and damping constants.

Webots includes a set of sensors and actuators frequently used in robotic experiments, e.g. proximity sensors, light sensors, touch sensors, GPS, accelerometers, cameras, emitters and receivers, servo motors (rotational & linear), position and force sensor, LEDs, grippers, gyros and compass. The robot controller programs can be written in C, C++, Java, Python and MATLAB. In our project we used C++ language.

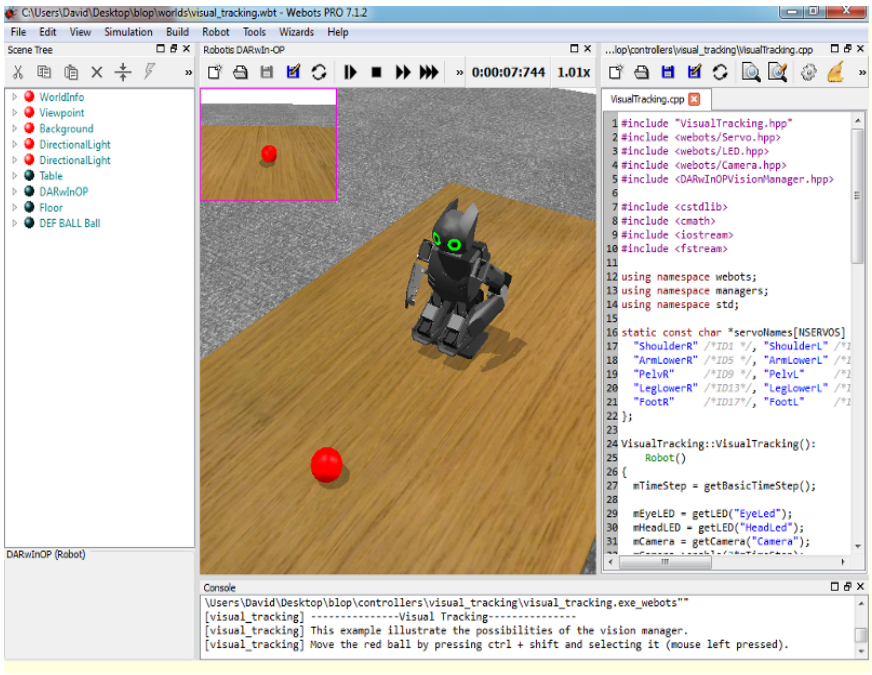


Figure 3: Simulation of a Robotis DARwIn-OP in Webots

Webot is not just a simulator, it is also possible to transfer a controller and objects previously developed in the simulation on to the real robot.

2.1.2 Features

- Fast robot prototyping
- Physics engines for realistic movements using ODE
- Realistic 3d rendering. Standard 3d modeling tools or third party tools can be used to build the environments.
- Dynamic robot bodies with scripting: C, C++, Perl, Python, Java, URBI, MATLAB
- Can simulate humanoid robots. For example: DARwIn-OP , Nao, , Fujitsu HOAP2, Kondo KHR-2HV, KHR-3, etc.

2.2 Technical information

Main Programming Language	C++
Formats support	WBT, VRML'97
Extensibility	Plugins (C++), API
External APIs	C++

2.3 The Robotics World

With Webots™, it is easy to create state-of the-art virtual environments for our robot simulations, using advanced graphics, with lights, shading, texture mapping, shadows, etc. Moreover, Webots allows us to import 3D models from most modeling software through the VRML97 standard.

Note: In our project, the hole part was prepared in CAD and transferred to Webot.

Our design robotics world *(Fig. 5)* consist of:

- Hole

- Red Pin
- Darwin Robot
- Floor

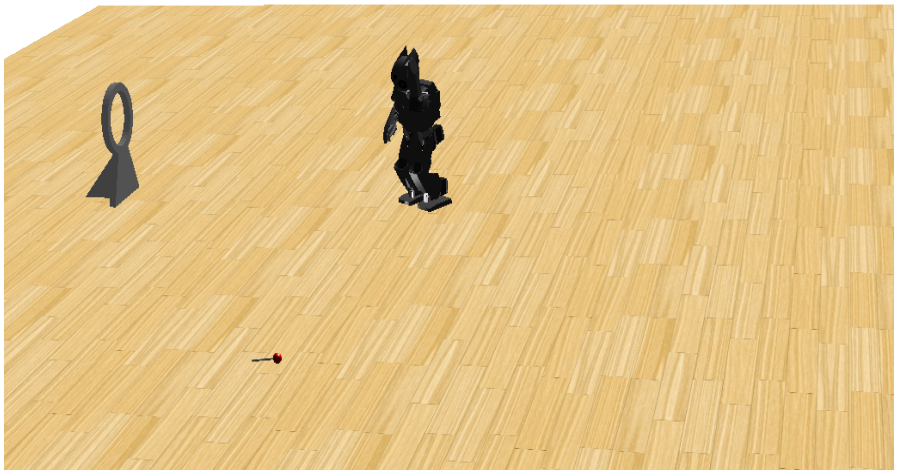


Figure 5: Robot World

2.4 Simulation with the control of Darwin Robot

The aim of this project was to fully integrate the DARwIn-OP in Webots. Webots is a simulator for mobile robots and DARwInOP is an open source miniature humanoid robot platform.
This integration has been divided in three main steps:

- Creation of a simulation model of the robot.
- Creation of a cross-compilation tool.
- Creation of a remote-control tool.

2.4.1 Simulation model

We tested our controller in simulation, without any risk of damaging the robot. We also can run automatically a lot of different simulations in a very small amount of time (to tune up parameters for example), which would be impossible to do with the real robot. The simulation model of DARwIn-OP is design to be as close as possible to the real one. It is equipped with the following sensors and actuators:

• 20 servos
• 5 LEDs (including 2 RGB ones)
• A 3 axes accelerometer
• A 3 axes gyroscope
• A camera

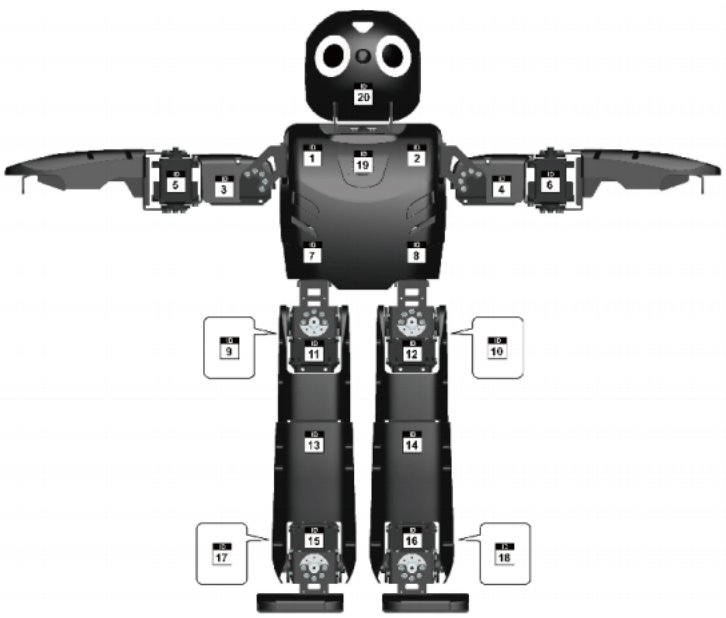


Figure 6: Position of the servos.

The following sensors/actuators are not present on the simulation model:
• The three buttons on the back of the robot are not present because they have no interest in the simulation.
• The microphones are not present in simulation because sound is not yet supported in Webots.
• The speakers are not present too because sound is not yet supported in Webots, but this will certainly be added soon.

2.4.2 Cross-compilation

A cross-compilation tool has been made in order to allow the use of controllers made in simulation on the real robot without any need of modifications. When our controller is doing fine in simulation, we will be able to send and run it on the real robot without changing anything to our code, just by pressing a button in the robot window.
To perform the Cross-compilation, send your controller to the real robot and make it run on it. This is done by going to the 'Transfer tab' of the robot window and perform followings steps.

- Set the connections settings:
  - IP address
    - ✓ Ethernet cable
    - ✓ Wifi connection
  - Username
  - Password
- Change the Makefile.darwin-op file
- Complete the "Robot Config" section of the config.ini file
  - Time step parameter
  - Camera resolution parameter

- Send a controller to the robot.
- Install a controller to the robot

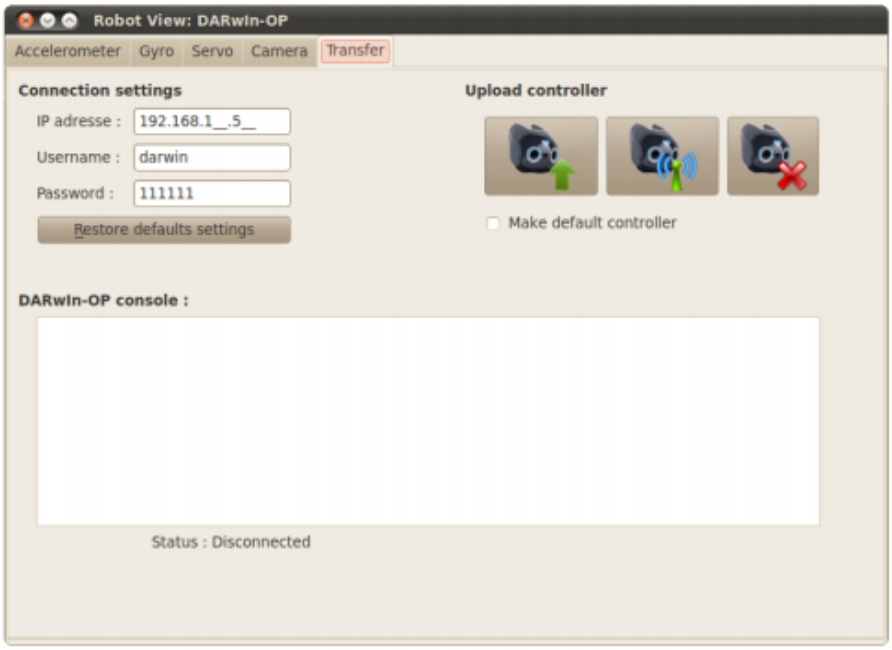


Figure 7: Transfer tab of the robot window

2.4.3 Remote-control

To debug or understand our controller's behavior, we will be able to see in real time the state of all the sensors and actuators on the computer screen. This is available both in simulation and on the real robot, and here again this is done in just one click. We will also be able to run our controller on the computer, but instead of sending commands to and reading sensor data from the simulated robot, it sends commands to and reads sensor data from the real robot.

Remote-control, is much simpler to use than cross-compilation, we do not set the time step in any files, or to edit any specific makefile, the exact same controller that in simulation can be used for remote control (without even having to recompile it). Moreover, the remote-control mode allows us to visualize the state of the sensors and actuators of the real robot in real time.

3 Software Development

3.1 Requirements

As previously discussed, the aim of this project was to get the Darwin-op robot to find a pin, pick it up and put it through a hole. In order to complete and develop this project extra specifications needed to be defined. This included assumptions needed and made for this project, for example the colour of the pin and the size of the hole. The full specification list can be found below.

3.1.1 Specs

- Assume pin is always on the floor (below robot arm level)
- End of the pin should be a distinct colour
- Assume the hole is at arm level
- Hole should be large enough for the Darwin-op robot arm to fit through it
- Pin is within 0.5-meter radius of the robot
- Hole is within 0.5-meter radius of the robot
- 30 min time limit to complete the task (battery length)
- Floor is a block colour i.e. not patterned
- Floor is a different colour from the pin
- Hole has a distinct outline
- Hole is vertical
- Assume ball is the same side of the hole as the robot
- Ball is always red (nothing else can be red)
- Hole always a circle (nothing else can be circular from eye level up)

3.2 Analysis

Once the specifications have been defined it was then possible to move on to the analysis. Here the key use-cases are defined including their description, preconditions and triggers.

3.2.1 Use Cases

Use Case Number	**1**
Use Case Name	Find the red pin
Use Case Description	The robot should scan its head up and down (eye level and below) while turning in an anticlockwise direction looking for the pin.
Use Case Preconditions	N/A
Use Case Trigger	Play Button is pressed

Use Case Number	**2**
Use Case Name	Walk to the red pin
Use Case Description	The robot should use a mixture of dynamics, control and trajectory planning to walk towards the pin. When walking towards the pin the robot should aim slightly to one side of the pin in in order to be positioned correctly to pick the pin up with the robots hand.
Use Case Preconditions	'Find the red pin'
Use Case Trigger	Once the pin has been found

Use Case Number	**3**
Use Case Name	Pick up / touch the pin
Use Case Description	The Robot must use kinematics in order to touch the pin. The robot needs to squat down and straighten its arm.
Use Case Preconditions	'Walk to the red pin'
Use Case Trigger	Once the robot is next to the pin

Use Case Number	**4**
Use Case Name	Find the circular hole
Use Case Description	As before, the robot should scan its head up and down (this time it should be eye level and above) while turning in a anticlockwise direction looking for the hole. It is proposed to use circle detection to find the hole. While colour detection could be used it is hoped the circle detection would make the hole finding more distinct.
Use Case Preconditions	'Touch the pin'
Use Case Trigger	After the pin has been touched

Use Case Number	**5**
Use Case Name	Walk to the hole
Use Case Description	The robot should use a mixture of dynamics, control and trajectory planning to walk towards the hole. When walking towards the hole the robot should aim for the centre of the hole.
Use Case Preconditions	'Find the circular hole'
Use Case Trigger	One the hole has been found

Use Case Number	**6**
Use Case Name	Put the pin through the hole
Use Case Description	The robot should straighten one or both of its arms and put the arm holding the pin through the hole.
Use Case Preconditions	'Walk to the hole'
Use Case Trigger	Once the Robot is at the hole

Use Case Number	**7**
Use Case Name	Check if fallen
Use Case Description	If the robot falls over on its front or its back it must recognise this and push itself back up to standing.
Use Case Preconditions	
Use Case Trigger	If the robot has fallen over

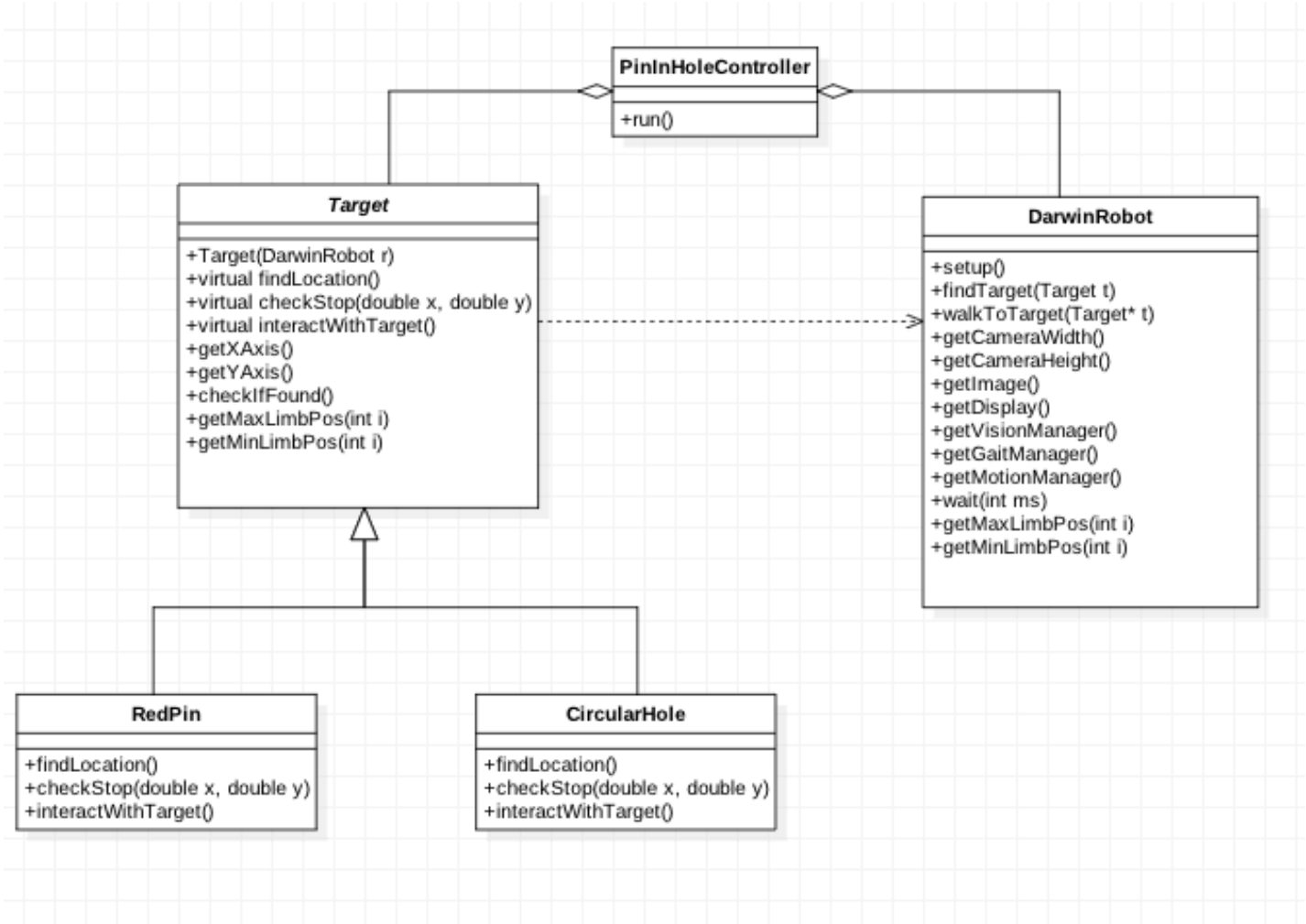
3.3 Design

Once the use cases were stated it was then possible to start designing the system. As previously mentioned the C++ program language was chosen as Webots supports this language and it allows for object orientated program to be used. The code was written in an object orientated design in order to provide modularity and reusability. It also allows for encapsulation allowing classes to hide and protect certain values.

Two main classes were defined; the robot class and the target class. Both the pin and the hole extend the Target class. Additional target items can be added, allowing the robot to find, walk

and interact with different targets in different manners. Details of these classes can be seen below in the class diagram.

3.3.1 Case Diagram



3.4 Implementation

As can be seen from the class diagram the software was split up into 5 different classes. The controller run method is shown below. The controller constructor method first creates an instance of the robot, pin and hole classes. The controller run method then calls each of the use-cases in turn. A call to the setup is done after finishing with the first item in order to make sure the Darwin-op is back to its walking state. Use case 7, 'Check if fallen', is called during each update to the timestep in order to make sure it is always checking that the Darwin is still up right.

```
void PinInHoleController::run()
{
   robot->findTarget(pin);          // use case 1
   robot->walkToTarget(pin);        // use case 2
   pin->interactWithTarget();       // use case 3

   robot->setup();
```

```
    robot->findTarget(hole);          // use case 4
    robot->walkToTarget(hole);        // use case 5
    hole->interactWithTarget();       // use case 6
}
```

Once all use cases are completed the destructor is called on each of the object in order to avoid any memory leaks.

As can be seen from the code above there are 3 main methods:

- findTarget
- walkToTarget
- interactWithTarget

Each of these methods have slight variations depending on what target the function is dealing with. A description of each of these methods and how they vary are explained below.

3.4.1 findTarget

When searching for the target the robot moves in a counter clockwise rotation while moving its head up and down. The range of head movement is limited depending on the target. Once the object is found the checkIfFound() method returns true and the robot can start walking towards the target.

3.4.1.1 Pin

The pin is found by using one of Darwin's pre defined methods in its vision manager. The vision manager is constructed with a specific colour hue, in this case red. The 'getBallCenter' method is then called on this manager. This method returns the x and y axis of where the center of the red object was found in the camera image.

3.4.1.2 Hole

While the same method for the pin could have been used for the hole it was decided to use the openCV circle detection method instead. The camera image first needs to be filtered before the circle detection is applied. By converting the image to grey scale and blurring the image it allows the circle detection to be more accurate and makes it easier to detect. Once the circle is detected *(Fig. 1* , small adjustments are needed to the x and y axis values in order to keep the circle focused in the robot's vision.

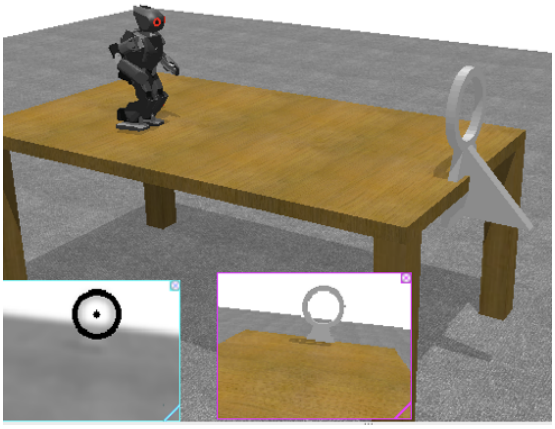


Figure 8: Circle detection applied on image

3.4.2 walkToTarget

Once the target has been found the robot starts to walk towards it given the specified x and y axis values. After each step the new relative x and y axis coordinates are then calculated and the step it repeated until the robot reaches the object. The stop is different depending on the target object.

3.4.2.1 Pin

As the robot moves closer to the pin it has to look further down towards the ground which in turn changes the y axis value. From this it is possible to gauge how far the robot is from the pin and stop when appropriate

3.4.2.2 Hole

Knowing when to stop at the hole is a bit more of a challenge. The robot moves towards the circle until it can no longer detect a circle (this is hopefully because the robot is too close to the hole). At this point it walks a set number of steps to get closer to the hole.

3.4.3 interactWithTarget

Different targets require different forms of interaction. When the robot arrives at the pin it should pick it up or touch it, however when the robot arrives at the hole it should put its arm through it.

3.4.3.1 Pin

Assuming that the robot has reached its target, the robot can then touch the target. This involves a combination of straightening out its arm to the side and bending its leg *(Fig. 2)*.

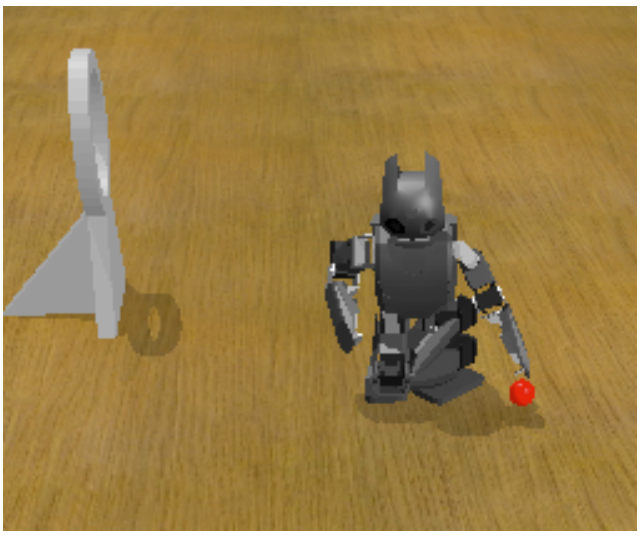


Figure 9: Darwin-op robot touching the 'pin'

3.4.3.2 Hole

Assuming the robot has reached its target the robot can then straighten out its arm in order to put its arm thought the target *(Fig. 3)*. In this case both arms are straightened as there is no control where the robot is relative to its target.

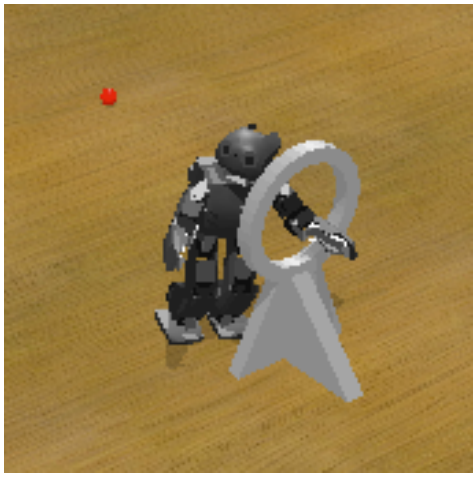


Figure 10: Darwin-op places its arm through the hole

3.5 Challenges

While writing this software there were a number of challenges and constraints. As can be seen, the arm that Darwin currently has does not allow for objects to be picked up. This was due to being unable to control the custom made gripper within Webots. As a results, when simulating this task, the robot touched the pin instead of picking it up.

Another issue was found when trying to get the Darwin to walk towards the hole. As explained, gray scale was needed in order to blur the image and get circle detection. However, when trying to do this directly in the CircularHole class the controller would crash. As a get around, the image was changed to gray scale in the robot class and then passed across. A number of issues were also found with finding and walking to the hole. The circle detection isn't that robust. Minor changes to both the x and y axis had to be made in order to keep the circle in the robot's

sight. Additionally, if the robot was at an angle to the hole, the hole would not be detected as it was viewed as an ellipse rather than a circle.

Another the main issues with the hole was knowing when the robot had reached the hole. As expected, as the robot got closer to the hole, the circle appeared bigger. Eventually, the circle became too large to fit in the camera so the robot was no longer able to detect it. Seeing as the Darwin-op does not have any distance sensors, at this point the robot is hardcoded to walk a set number of steps until it reaches the hole.

3.6 Further Work

While the project managed to achieve a number of its goals there are a number of improvements that could be implemented in future work.

As discussed, we were unable control the custom gripper within Webots. If this was possible then the methods in the redPin class would need to change from touching the pin to picking up the pin.

Another main issue was detecting and know when to stop at the hole. OpenCV circle detection was used but there may be other alternatives that allow the robot to navigate to the hole better. Currently the robot putting its hand through the hole is not stable as it is chance as to whether the robot put its left hand, right and or no hand through the hole i.e. straightens its hands either side of the hole. Ideally the robot should put the same hand that picked up the ball through the hole in order to put the ball through the hole. As the code is modular it is possible to improve the code for the Hole without effecting the robot finding the pin.

One last improvement would be to update the pin class in order to make the changing of the pin easier. By changing the vision manager values it should be possible to enter any colour of pin for the robot to find.

4 Design

The purpose of this chapter is to tackle the issue of designing a new gripper, or at least, a part that would sustain a gripper which could then be used for the objective at hand. Here will address our choices for software, the assumptions that were made in our design decisions, the process through which we designed and manufactured the part and the final result itself.

4.1 CAD Software

Of the many choices available to us for designing a 3D model, our decision fell on Solidworks for 3 main reasons:

- Productivity
    - Intuitive 3D design, with a focus on innovation.
    - Built-in intelligence that creates a user friendly environment and accelerates the design process.
    - Free Student Licence.

- Power
    - Creates 2D drawings faster and almost automatically, ready for workshop prototyping.
    - Speedy design ensures accuracy with focused industry tools and terminology.
    - Built-in Finite Element Analysis allows for real world simulation.
    - Allows not only for part design, but also environment creation.

- Community
    - Due to its widespread applications, there is an active and still growing user community to which we could connect, share, and discuss any issue.
    - New talent is being drawn to this tool, which ramps up its usage and innovation techniques used for 3D design.
    - An accessible network of resources, people and ideas.

Even though all the above provided quite a few reasons for us to choose this platform, let us bear in mind that one of the most important issues would be the ease to go from design to the workshop, which, due to its in-built 2D drawing engine, Solidworks is able to tackle.

Such importance derives from the fact that the prototyping process, as we were able to experience in the course of this project, is often iterative, since small adjustments are always needed to bridge the reality gap between design and the actual final part.

In next sections we show how we built the new part as well as the gripper we used and how this could be connected to the existing darwing-OP model. We will present, not only the deigned part in 3D, but also its drawing and the result we obtained.

4.2 Process

This section focus on the various steps of our design process which can be further divided into 3 different categories:

- Measurements: Here, we focused on getting acquainted with the existing model of the Darwin-OP itself and the purpose of this step is to gather accurate measures of those parts where Darwing and our designed part will connect.
- 3D/2D Modelling: Here we explore how can a new part be built, and which shape must it take so that it works as intend both in respect to the connection to Darwin as well as the connection to the gripper.
- Manufacturing: This point aims to give an overview of our choices for manufacture the part we needed and the challenges we have faced, both technical and not in order to do so.

4.2.1 Measurements

Getting the correct measurements for building a new part is a crucial step in any design project. At this stage we focused on, not only getting acquainted with the more generic dimensions of the

Darwin-OP shown in Figure 11, but also its specific structure shown in Figure 12.

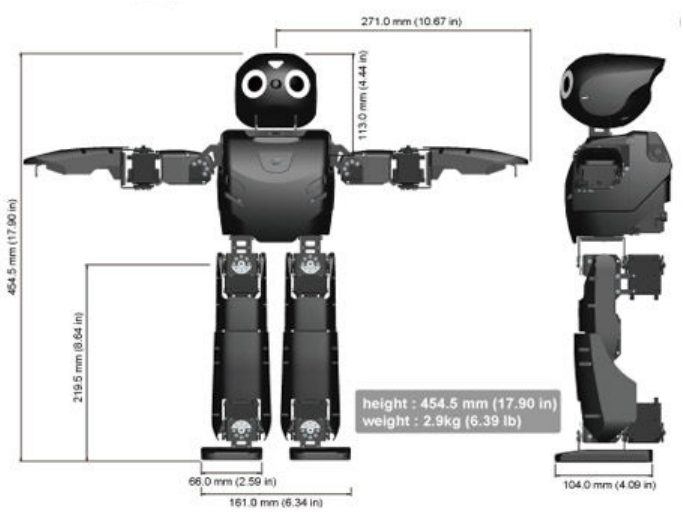
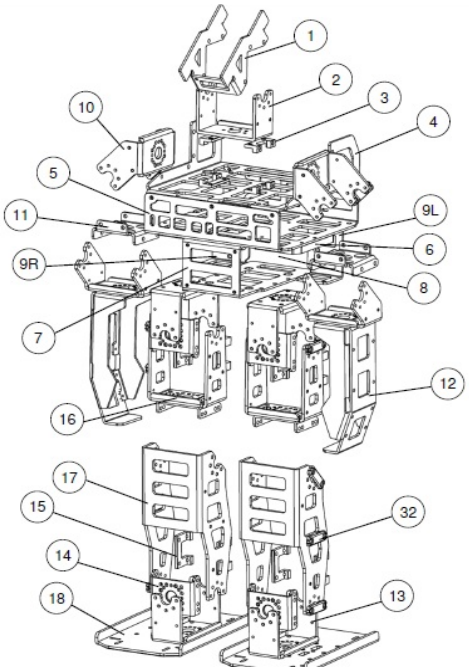


Figure 11: External Measures



Part Number	Description
1	Head Bracket
2	Neck Bracket
3	Chest Actuator Connectors
4	Upper Chest Bracket
5	Lower Chest Bracket
6	Mounting Bracket
7	Battery Bracket
8	Battery Cover Bracket
9L	Back Left Pelvic Bracket
9R	Back Right Pelvic Bracket
10	Angled Actuator Bracket
11	Arm Actuator Mount
12	Hand
13	U-Actuator Bracket
14	Leg Actuator Mount
15	Actuator Connector
16	Thigh Bracket
17	Knee Bracket
18	Foot
32	Cable Holder

Figure 122: Components

Looking closely at the second Figure 12 we realized that due its inherent open-source structure, the part we would have to replace would be number 12 (the Hand) and that the part our design should connect to would one of the various “Angled Actuator Brakets” listed as number 10.

Figure 13 shows some early sketches of the part we aimed to design.

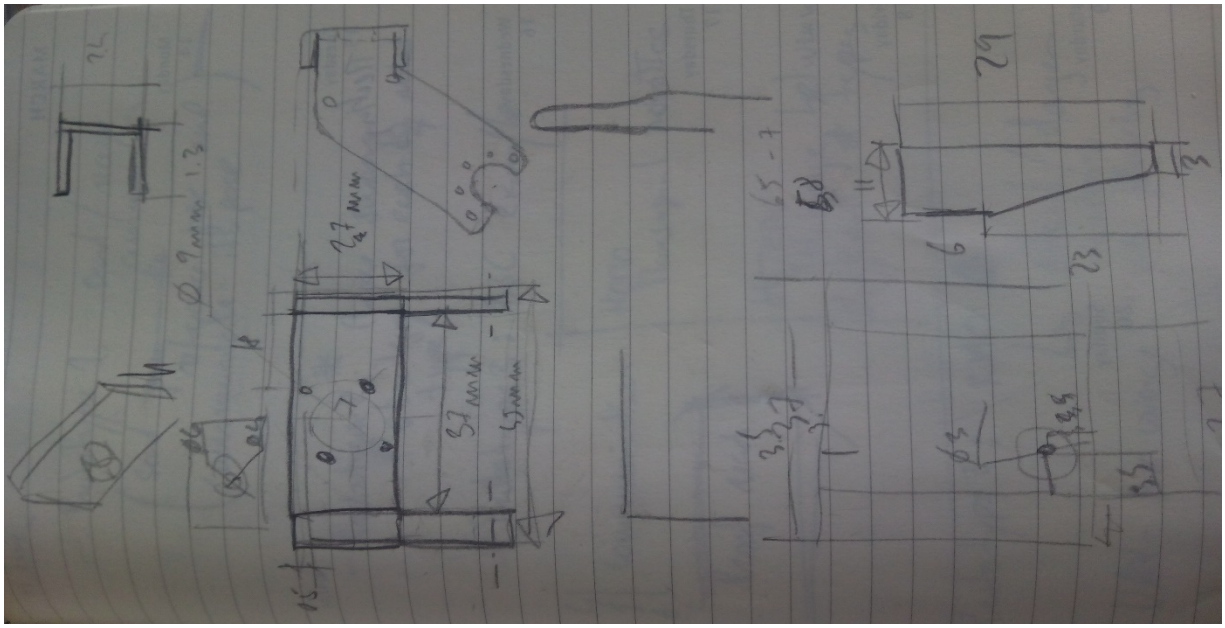


Figure 13: Early Sketch

4.2.2 3D/2D Modelling

Stepping from the sketching phase into the 3D modelling we started by building the part which would replace the original hand and hold the gripper. We also built the gripper in the 3D model so that we could create an assembly which aimed at mitigating any mistake in our measurements thus facilitating the next phase of our project.

Below we show two figures of the final assembly, one with all the parts together and an exploded view of the same showing the different parts that constitute this model.

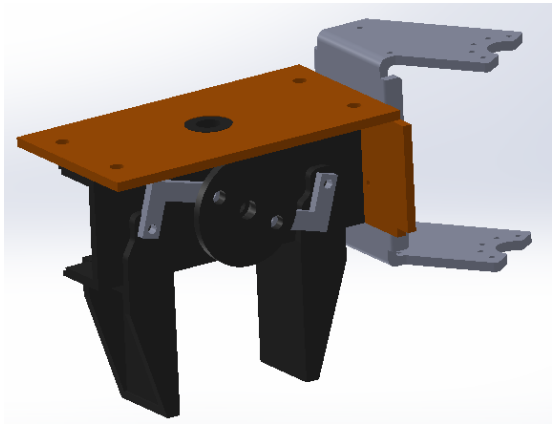


Figure 15: Assembly

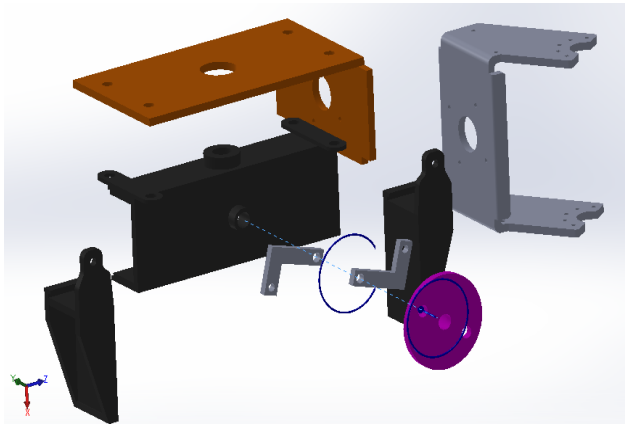


Figure 145: Exploded View

Note that the part in question is that of a brownish colour, whereas the black parts represent the gripper we were provided with and the grey part represents the "Angled Actuator Braket" mentioned above.

Let us note how we opted for a simple design, yet one that would fulfil all the specifications needed i.e.:

- Full connectivity to Darwin's main structure, through threaded holes (M1)
- Full connectivity to the provided Gripper, also through threaded holes (M3)

- A shape that allows for cable passing, necessary for driving the actuator itself.
- Structural integrity.

At this stage, getting the part ready for the workshop implies providing the 2D drawing. Figure 16 shows the exact dimensions and shape of the part we designed.

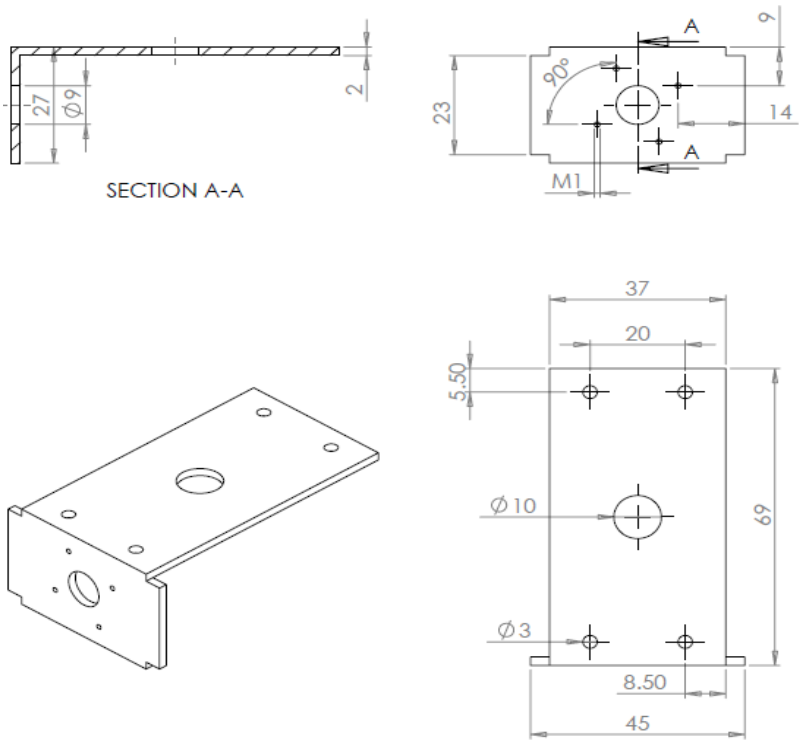


Figure 16: Exact Dimensions

4.2.3 Manufacturing

As mentioned above, one of the key features to take into account when designing a mechanical part is structural integrity. Nevertheless, as in our case, we aim to build a prototype that would demonstrate a proof-of-concept work (and not a fully marketable product) we decided to make use of the most common materials and manufacturing techniques, without compromising the goal of our project. For that reason, this part was built using a 2mm thick aluminium plate, with thread holes (M1 and M3) where the "L" shape was achieved by bending a strait plate. Figure 17 shows the intended result.

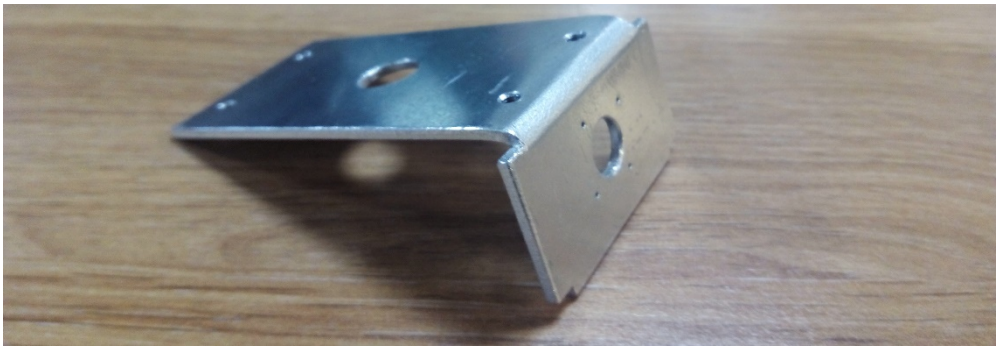


Figure 17: Manufactured Part

5 Electronics – Custom Intelligent Servo

When designing a robotic gripper for Darwin, the first idea was to use the same motor as Darwin already uses for its motions since Darwin already provides control and power over this type of motors. Darwin uses so called “intelligent servos” which are explained in the next section.

Figure 18: TODO

5.1 Introduction to Intelligent Servos (referred as IS)

5.1.1 Definition

Intelligent Servos are a new breed of servos used in high precision applications. This include applications that require exact positioning through Kinematics. An example application is a human like robot balancing on one lag which is a multi-disciplinary challenge. Nearly all current robots are using such “intelligent servos”, some with modifications that make them proprietary.

5.1.2 MCU Inside

The first question to ask is what’s inside an Intelligent Servo that differs it from a usual servo. The “intelligent” of the servo already reveals the most important part, the intelligent servo has a brain / micro controller (MCU) that provides the communication and feedback of the position measurement. The measurement itself is carried out by an encoder.

5.1.2.1 The case of “Dynamixel MX-28”

As described in the previous section, IS’s are characterized by its MCU. The MX-28 uses an ARM CORTEX M3 based MCU for the processing. The precise measurement is provided by a magnetic encoder which also stands for high durability of the measurement device (is often identified as a source of errors).

5.1.3 Communication

How do Intelligent Servos communicate? Hence the IS has its own microcontroller this demands for a communication using a protocol. Cheaper servos are usually driven directly using PWM. The intelligent servos are typically using either RS485 or RS232 as interface. Since there is no standard, the communication itself strongly depends on the implementation of the manufacturer. In the case of Darwin and specifically the “Dynamixel MX-28” a proprietary protocol is used. It has similarities to a token-ring communication with a closed loop. A typical communication process is demonstrated in the picture below:

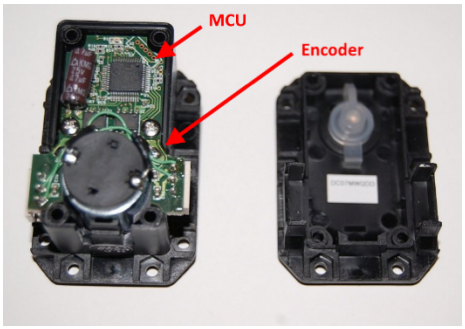


Figure 19: TODO

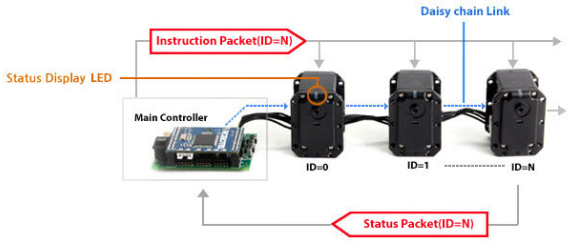


Figure 20: TODO

5.1.4 Key Features

The key features of the Dynamixel MX-28 Intelligent Servo Summarized:

MCU:	ARM Cortex M3 manufactured by SI (72 MHz)
Encoder:	12-bit magnetic encoder
Running Degree:	0-360° or continuous
Motor:	Maxon RE-max Metal Brush (PN:214897)
Link:	TTL

Having all that said it's no wonder that the costs of an intelligent servo is usually 10 times higher than the cost of an ordinary PWM controlled servo. In the case of Darwin's Dynamixel MX-28 Servo it exceeds the available budget. Therefore, a different solution needs to be invented / investigated. Since a servo requires power and some sort of control the Idea was to create our own Intelligent Servo which is described in the next section.

5.2 Custom Intelligent Servo

Our approach to an intelligent servo was to rebuild the environment of an intelligent servo as "adapter" to a cheap PWM stepper motor.

5.2.1 Requirement Analysis

Since all servos are connected in serial, the power and communication is provided by the antecedent servo. The communication of the MX-28 is using TTL serial hence our servo should be able to communicate using this protocol. Resulting an MCU/IC is a mandatory part. The required power for both the MCU and the Stepper Motor is provided by the previous servo.

Unfortunately, the provided power is at 12 Volt and a stall current of 1.4 Ampere. Since our stepper motor is working best between 5-6 Volts an AC-AC Voltage Regulator is required. In addition to that an MCU typically runs at 3-5 Volt which would require an additional Voltage Regulator just for the MCU.

5.2.2 Intelligent Servo - Our Approach

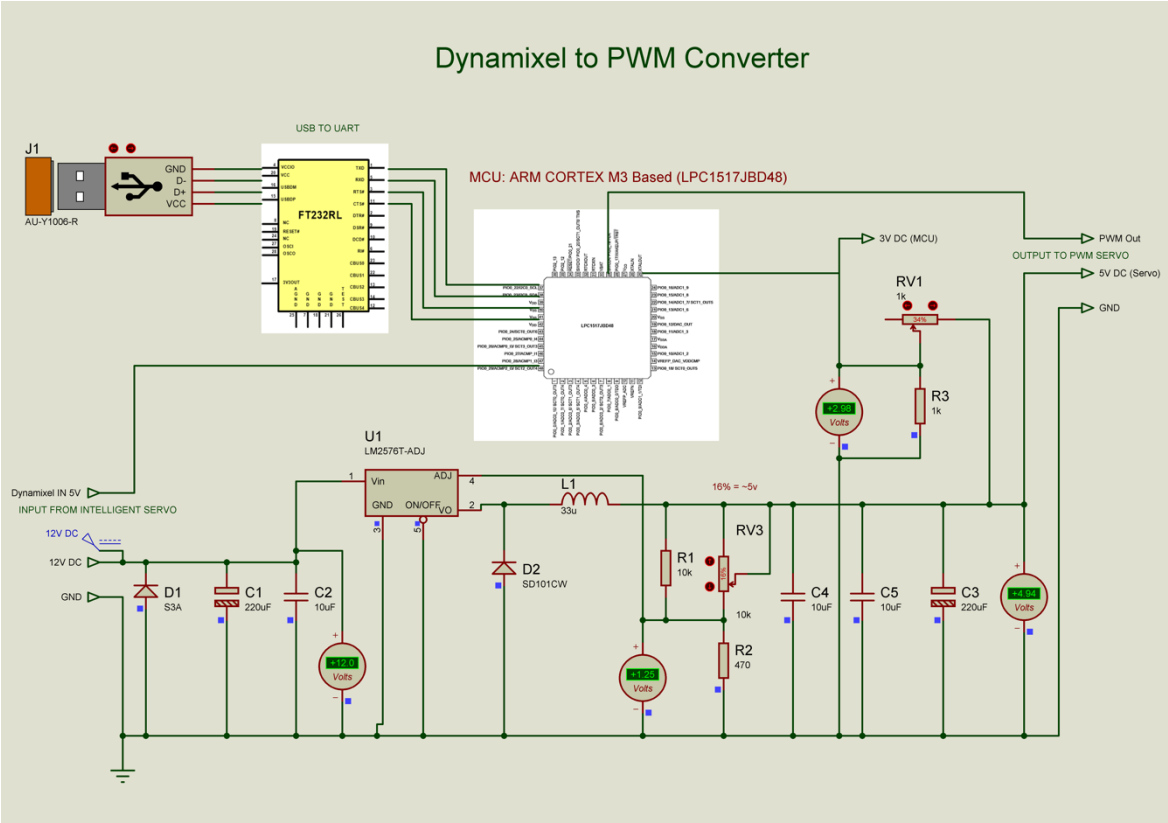


Figure 21: TODO

In our approach we used a Linear Voltage Regulator to “step down” to 5 Volt for the Servo. The used MCU in this case requires a 3 Volt input, hence we “step down” using regular resistors. The MCU itself is similar to the MX-28, ARM Cortex M3 based. For an easy access and programming interface to the MCU and FT232RL IC is used. This enables us to directly interface the MCU using USB. The diodes and capacitors are for the protection and are – theoretically – already implemented in the Voltage Regulator and in the Servo. Since we want to make sure that Darwin can’t be damaged and its “best practice” we have added them additionally to our Circuit.

5.2.3 Critical Evaluation of the Custom Intelligent Servo Approach

What would we make different now? The first thing that was brought to our attention is the inefficiency of linear voltage regulators. These should be replaced by so called switching voltage regulators that emit less heat and are more efficient. In addition, the used Voltage Divider (the resistors) should be replaced by another Voltage regulator IC. This would save additional power which is a very important factor for a battery driven robot. If more time and budget would have been available, the stepper motor should have been replaced by a professional motor similar to the one used by Darwin.

5.2.4 Learning Outcome

This approach helped a lot in understanding of modern servo technology and why its currently on the rise. Especially in high precision applications where a lot of Kinematics is involved these servos are frequently used. In addition to that we learned about the Electronics required to drive these intelligent Servos. Furthermore, we now understand better what happens inside a humanoid robot like Darwin and why they are so expensive.

5.2.5 Reality

After understanding how intelligent servos work and completing our own design we realized that the "nice" approach is not realizable in-time anymore. Resulting we had to create a simplified version of the above concept. This simplified version is explained in the next section.

6 Simplified Intelligent Servo

6.1 Selected Servo

The motor which was given to us was Servo motor **HS-422** Standard made by Hitech. It is very simple and cheap.

Figure 22: TODO

Some of its features are

Parameters	Description
Control System	+Pulse Width Control 1500usec Neutral
Operating Voltage	4.8-6.0 Volts
Torque	45.82/56.93 oz-in. (4.8/6.0V)
Direction	Clockwise/ Pulse Traveling 1500-1900usec
Current Drain (4.8V)	8mA/idle and 150mA no load operating
Current Drain (6.0V	8.8mA/idle and 180mA no load operating
Motor Type	3 Pole Ferrite

6.1.1 Design of Power Supply

Operating voltage of servo is 4.8-6 V and Arduino board is also around 5- 12V . But voltage coming from Darwin is a 12V . So need arises to step down voltage . Voltage regulators circuits is used to step down voltage from 12 to 5v.
The circuits which can be used for step down voltages are

- Linear voltage regulators circuits
- Switching voltage regulator circuit

Linear voltage regulators circuit was used in our project since Switching circuits were not available in university. But Switching voltage regulator circuit are more suitable for our project since they are more efficient than linear voltage regulators circuits.

6.1.2 Linear voltage regulator design

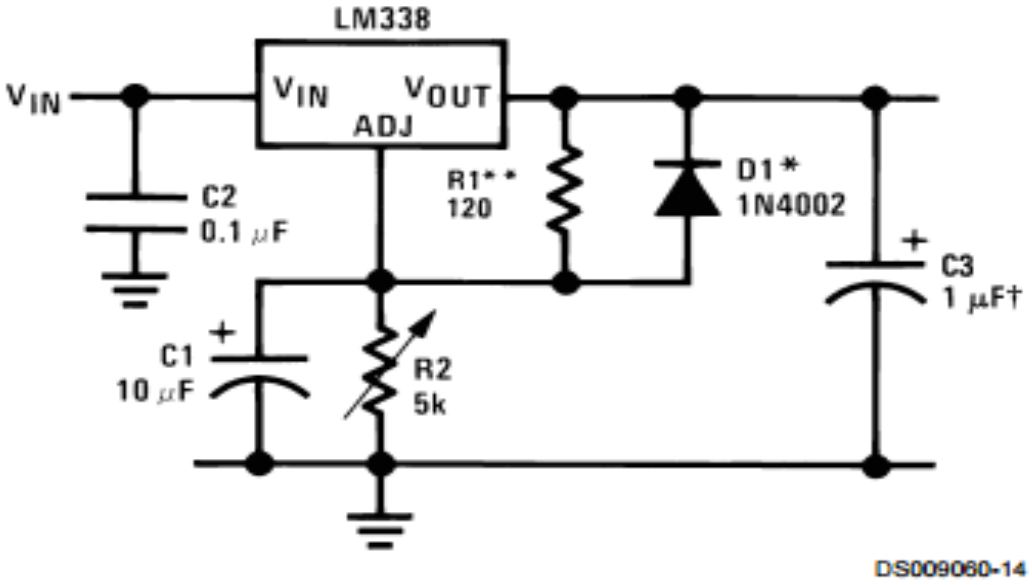


Figure 23: TODO

$$V_{OUT} = V_{REF}\left(1 + \frac{R2}{R1}\right) + I_{ADJ}R2.$$

Input is 12 v and output is 5v which is given to servo and Arduino. Filters circuits(capacitors) and diodes are used to protect Darwin motors from receiving any backward current from the circuits.

6.1.3 Servo Movement

The servo is controlled using PWM Technique. The term pulse width modulation refers to the technique of varying a signal's pulse width to control a device such as a servo. Used in many applications apart from this such as Lamp dimmers, motor speed control, power supplies, noise making due to efficiency and simplicity of PWM signal as well as flexibility of pulse modulated waveform. The clock cycle, duty cycle, amplitude are some of basic parameters of PWM. Shown above is the simple PWM signal.

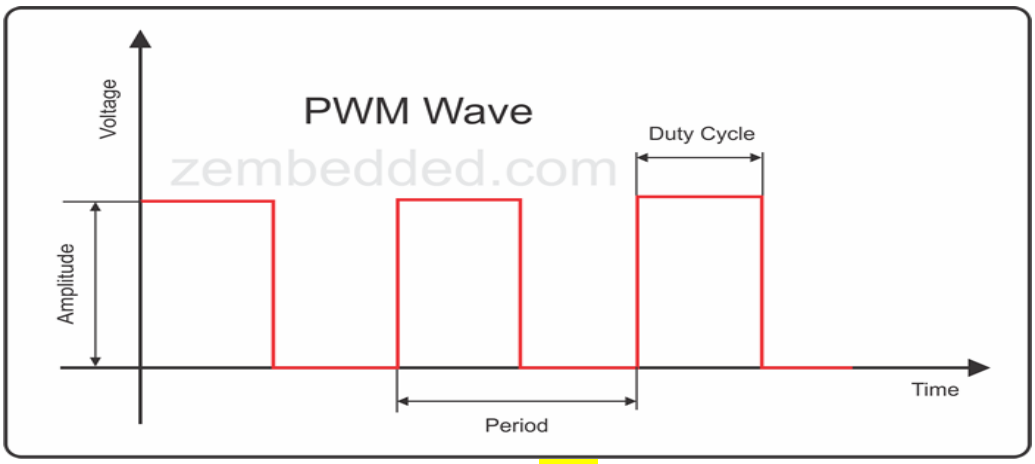
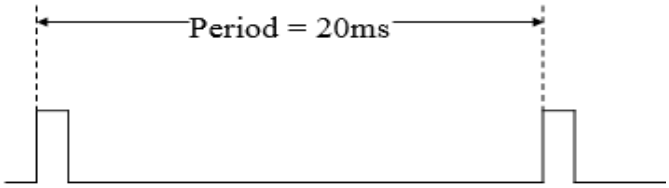


Figure 24: TODO

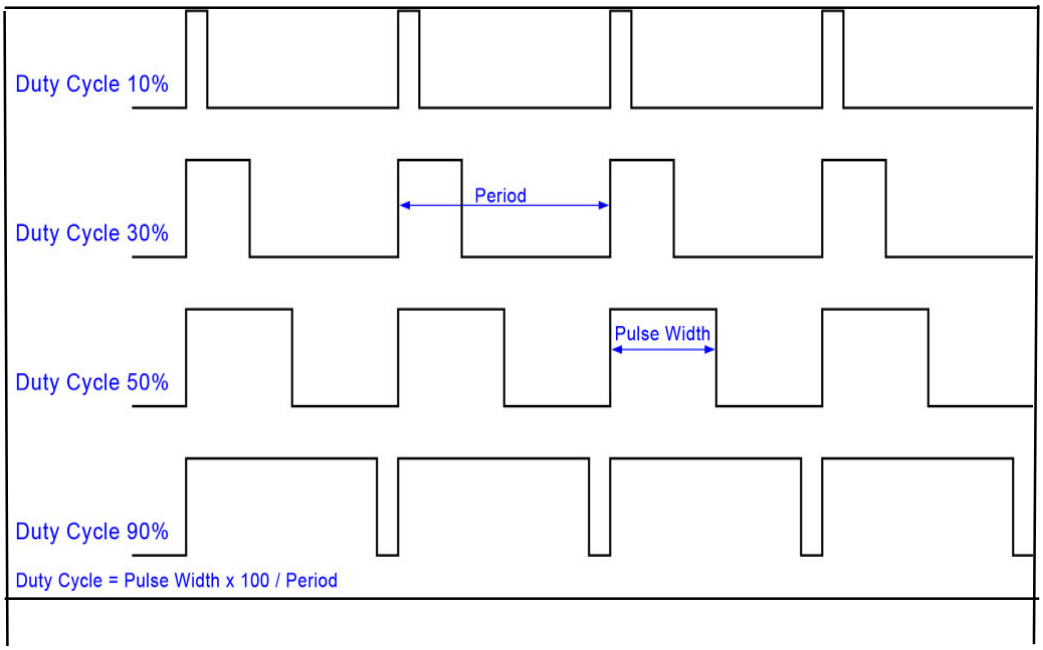
a)

The first parameter which is a clock cycle is the frequency of signal measured in hertz and easy to understand. The servo's control signal is a 50 Hz pulse train.



b)

The other parameter which is a duty cycle involves switching of a signal. It is explained with waveform as shown above



All three signals shown above are square wave oscillations modulated as per their oscillation width, so called duty cycle and have same frequency. The parameter which is changed many times during program execution is the duty cycle. The frequency remains same but signals differ in pulse width. Duty cycle controls amount of power supplied to external components.

6.1.4 Servo Rotation Principle

The shaft of servo is rotated by passing a PWM signal on a yellow wire as shown in fig above. The servo maintains the angular position of the shaft as long as signal exists on its input line and position of shaft changes, if signal given to control(yellow) wire changes.

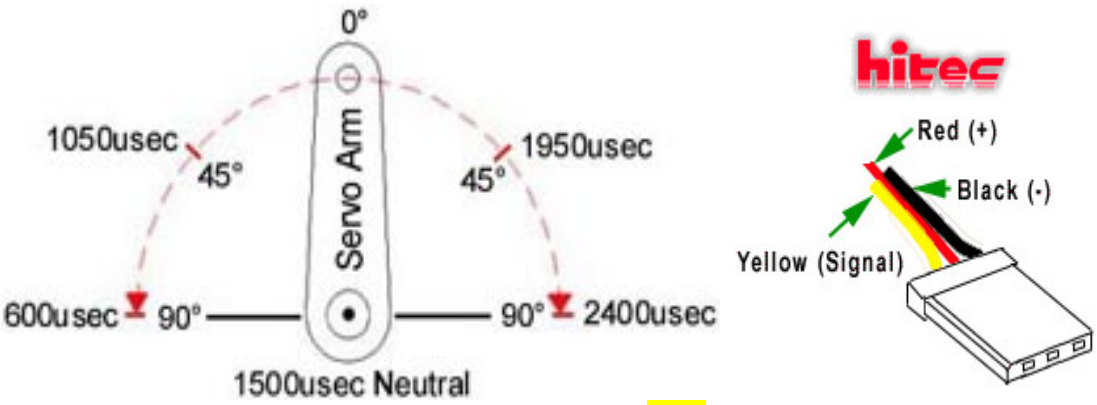


Figure 25: TODO

A control wire communicates the desired angular movement. The control signal's **pulse width** determines the shaft's angle of rotation. The minimal width and the maximum width of pulse that will command the servo to turn to a valid position are functions of each servo. Even different servos of the same or different brand, will have different maximum and minimums.
In our servo, the pulse width ranges from about 0.6ms to about 2.4 ms as seen in fig[a] and can be interpreted as follows:

- Pulse width of 1.5 ms will give 0° rotation.
- Pulse width of less than 1.5 ms will give counter clockwise rotation up to 90°.
- Pulse width of greater than 1.5 ms will give clockwise rotation up to 90°.

Depending upon signal received by servo from Arduino platform, the servo rotates in desired directions.

6.1.5 Generation of pwm using Arduino platform

We can generate PWM in Arduino platform using PWM inbuilt technique or manual method.

We used manual method since the built in PWM frequency did not matched the servos expected pulse timing.

In Manual method, PWM is generated in Arduino platform by writing code as follows

```
#include <Servo.h>
Servo myservo;                          // create servo object to control a servo
intpos = 0;                             // variable to store the servo position

void setup()
{
myservo.attach(9);                      // attaches the servo on pin 9 to the servo object
}
void loop()
{
myservo.attach(9);

for(pos = 0; pos< 180; pos += 1)        // goes from 0 degrees to 180 degrees
 {
myservo.write(pos);
delay(15);                              // waits 15ms for the servo to reach the position
 }

for(pos = 180; pos>=1; pos-=1)          // goes from 180 degrees to 0 degrees
 {
myservo.write(pos);                     // tell servo to go to position in variable 'pos'
delay(15);
 }

myservo.detach();          //Detach the servo if you are not controling it for a while
delay(2000);
}
```

Figure 26: Standard Servo Rotation to Exact Angel code

7 Conclusion and discussion

In the course of this project we have faced many constraints and challenges that incur from undertaking a project with such a practical approach and objective. These can be divided into 5 different, yet related, categories, namely:

- **Set a fixed set of specs from the beginning –** The fact that we were given total freedom on how to work out the problem and tackle its many facets, came as the first big challenge we faced as group. This implied going over the problem in an iterative manner, constantly reassessing our assumptions as we learned more about the Darwin-OP, its functionalities and electro-mechanical structures. At this point the concise and concrete definition of the problem was made by writing a list of specifications we aimed to achieve during the course of this project.

- **Project plan should have been in place –** A direct consequence of the above, resulted in the fact that also our assumptions, for how much time a certain task would take, were often, inaccurate. Despite the fact our knowledge of the platform increased with each passing week, our starting point was from total inexperience, which we were able to gradually overcome more efficiently as a result of both trial and error and research and by keeping a methodical approach. Nevertheless, the natural initial misjudgments of time consumption were a critical point for this (and most) project(s).

- **Tasks could have been split up with more defined roles –** Naturally, as in any project, our knowledge of the problem increased with the passing of time we dedicated to it. In retrospective this caused us to divide tasks amongst ourselves not always in the most efficient manner. Once

more, as our familiarity with Darwin-OP improved so did our judgment on how tasks could be split and so our effectiveness also increased.

- **Design the world in detail before writing the controller –** As mentioned, many of these issues are very entwined and dependent of each other. Building the simulated environment and coding the controller are naturally no exception. Even though at first we assumed that there was some independence between the two, and so we could (for example) start coding the methods that would make the Darwin move towards a generic target or grab an object, at a later stage we were confronted with the fact that our generic assumptions, regarding the environment, had to be further developed so we could progress with code implementation. This also falls under the previous point where we mentioned that division of tasks in an effective manner was highly dependent on our knowledge of the problem itself and all the hardware and software capabilities.

- **Hardware should have been designed earlier in order to have time for modifications and testing –** This is the bullet point which is definitely more concerned with the execution of ad hoc mechanical parts and electrical components needed for this project. Availability of resources, either they are materials, workshop schedules or defective components are a constant in any real-life project. In this particular case, and once more, as our understanding of the task deepened, we may say that, in retrospective, we could have started building the extra components earlier. Nevertheless, we felt that as a group it would be most advantageous for everyone if each member was comfortable in discussing any part of the project, and for that reason we purposefully focused on learning from each other's expertise.

To sum up, we believe we came quite close to our initial goal which was having the real Darwin grasp a pin and insert it into a hole, namely for the following reasons:

- We have showed we were able to use form and colour detection.
- We have showed we could command Darwin's movements precisely, given the task.
- We have been able to build a mechanical component which supports a different gripper and connects to Darwin's main structure
- We've shown we could use Darwin's power supply and communication protocol to actuate a separate gripper.

Despite the fact the final step is, obviously, still to overcome, we are confident that with this project we have set a solid foundation in terms of both simulation and electro-mechanical design so that future students can continue our work without the need to tackle the most low-level challenges.

8 References

Ha, I., Tamura, Y. and Asama, H. (2013). Development of open platform humanoid robot DARwIn-OP. *Advanced Robotics*, 27(3), pp.223-232.

Williams, K. (2004). *Build your own humanoid robot*. New York: McGraw-Hill.

Cyberbotics Ltd. Webots: robot simulator – overwiew [cited 2011 June 19]. Available from: http://www.cyberbotics.com/overview

Kazuhito Yokoi, Fumio Kanehiro, Kenji Kaneko, Shuuji Kajita, Kiyoshi Fujiwara and Hirohisa Hirukawa. The International Journal of Robotics Research 2004; 23; 351. Experimental Study of Humanoid Robot HRP-1S

Serena Ivaldi , Jan Peters , Vincent Padois and Francesco Nori : Tools for simulating humanoid robot dynamics: a survey based on user feedback Available from: http://www.ausy.tu-darmstadt.de/uploads/Site/EditPublication/ivaldi2014simulators.pdf

9 Individual Contributions

Hugo: I took the necessary measures to build the new gripper support as well as developed the CAD model and scheduling the manufacturing of it in the workshop.

Markus

Samir : I completed integration of simulation between Darwin and Webot ,also developed world part of project.

Waqar

Roshenac: I designed, programmed and commented the simulator code in order to make the Darwin walk towards the ball, touch it, find the hole and put its hand trough the hole. I also wrote the chapter on software development and put all the different parts of the report together.

Shakh- Izat: I gathered information about Darwin-Op and its specifications. Also I designed presentation.