Robocup
Detailed Design Report

Group 4

Hayden Watson – 26781242

Luka Whitaker - 48420886

Conner Horn - 36757479

ENMT301

Mechatronics System Design

2025

University of Canterbury

Contents

Introduction

This report aims to outline the progress and developments made to achieving the specifications for the Robocup competition. It covers the current state and the planned tasks to make it ready to compete. Currently, the robot is partially finished, with the key focus being the pickup and motion of the robot, as well as its general structure and motion. Key areas to make the robot competition ready include the implementation of a diversion gate for the fake weights and a navigational algorithm.

The robot is designed with a combine harvester as its pickup method, which uses a ‘crisscross’ pattern of rubber bands to direct the weights and provide appropriate tension. An aspect that was changed from the concept development was the use of two wheels to ensure the pickup ramp could traverse low lying bumps and provide an appropriate height to the ramp. Other aspects of the robot include weight detection sensors and a slider ramp for the weights to travel down to the back gate. These weight detection sensors currently are used for wall detection and will need to be programmed to detect weights. The ramp and back gate are currently functional but will rely on future components to be fully operational.

Future iterations of the robot will need to incorporate a diverter and inductive sensor, to determine whether the weights are desired. This will include a servo, to push the weight out of the robot or into the storage area. For the navigation algorithm and wall detection, the IMU, a colour sensor and proximity sensor will need to be included. It was deemed that the use of the measurement wheel was going to have greater inaccuracies than that of an IMU. The IMU also had multiple types of data that could be used such as the heading angle. Hence, it was decided that the robot should use the IMU instead. The colour sensor will determine when the robot can drop the weights at the home base. A third sensor, mounted on the front of the robot, will be required to ensure no front on collisions, with walls that cannot be seen by the weight detection sensors. Code will need to be developed, which will have a focus on mapping the arena. Key ideas on the algorithm will be to travel to locations that are unexplored, and to return home after the 90 second mark.

Progress to Date

Collection Mechanism

Rubber bands were used on the barrel of the combine harvester to catch and secure the weights. Through testing, several changes were made to the rubber band selection and layout for reliability. The initial design featured rubber bands stretched across the barrel in a linear layout, using 3 mm wide bands. However, this configuration in conjunction with the 3mm rubber bands lacked the tension required to consistently pick up weights during testing.

To overcome this issue, the band layout was changed to a crisscross pattern as seen in Figure 1. This new setup provided more tension in the rubber bands as well as a small point of contact directly in the center of the bands to reliably secure the weights. In addition to this, the thickness of the rubber bands increased to 10 mm to improve both tension and durability. The combination of these changes resulted in a more reliable and durable collection system.

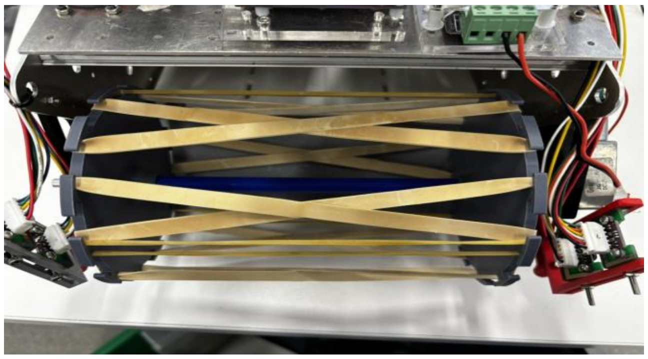

Figure 1 - Rubber Band layout

The harvester has remained the same from concept design. A wheel with 12 teeth to secure the rubber bands was designed. The reason for 12 teeth meant that a large range of symmetrical rubber band layouts could be tested, as 12 splits into groups of 2, 3, 4 and 6. A protrusion was included on the wheel, with a 6.5mm hole to fit a shaft and a grub screw to secure it. A full engineering drawing can be found in Appendix A.

The harvester runs off the 10 rpm motor, as this was determined through some simple testing, to have sufficient torque. A large torque was required due to the significant height of the ramp, something the 50 and 100 rpm motors could not provide.

A front-mounted ramp was developed alongside the combine harvester to guide the weights up and into the robot for sorting and storage. This went through multiple iterations during the design process. The first iteration, depicted in Figure 1, used a fixed, solid ramp with minimal clearance to the ground. Despite allowing weights to be easily guided into the robot, it introduced a problem to overcome. This problem being that the ramp was prone to catching on obstacles, such as bumps and ramps. This resulted in a negative impact on the robot’s mobility.

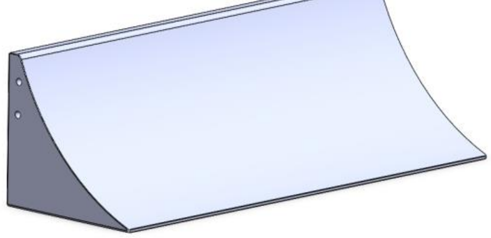

Figure 2 - Ramp Iteration 1

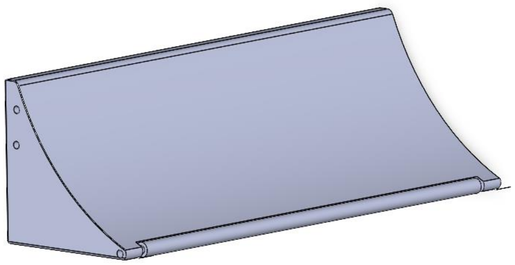

Figure 3 - Ramp Iteration 2

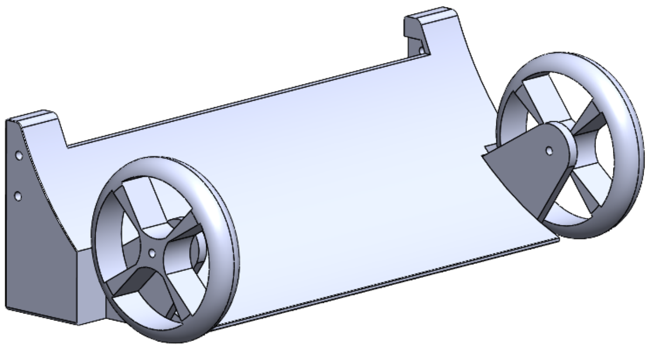

Figure 4 - Ramp Iteration 3 (final design)

In an attempt to overcome this, a roller was added to the front of the ramp in the second iteration (Figure 2) to assist in lifting the robot over obstacles. This, however, created a new challenge: finding the right roller radius. A large radius allowed obstacle clearance but made it more difficult for weights to transition from the ground onto the ramp. In contrast, a smaller radius still allowed reliable weight collection but was unable to provide sufficient obstacle clearance. This was determined to not be the solution, but a step in the right direction.

The final design (Figure 4) combines the strengths of earlier iterations. A low-profile ramp was kept close to the ground for reliable weight pickup, while wheels are mounted just in front of the ramp, outside the combine. These wheels meet the obstacles before the ramp does. In turn, lifting the ramp before it hits the obstacle. Once the lip of the ramp has cleared off the obstacle, the tracks are able to push the robot forward and proceed with normal navigation and movement. This design maintained reliable weight collection while allowing the robot to traverse the arena regardless of the obstacle before it.

The harvester was then tested with weights in different orientations such as, standing upright, parallel to the harvester and perpendicular to the harvester. Figures 5 to 7 show the testing set ups and Table 1 shows the results, with full testing found in Appendix B.

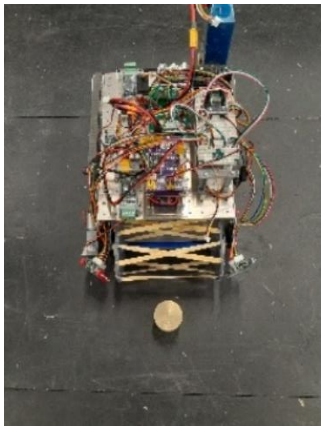
Figure 5 - Upright weight testing

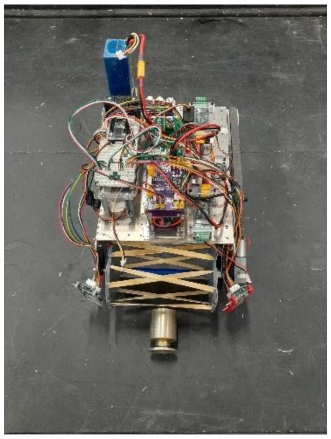
Figure 6 - Perpendicular weight testing

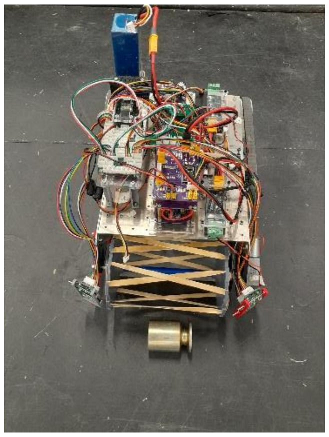
Figure 7 Parallel weight testing

Table 1: Weight orientation success rate in pickup

Weight Orientation	0.5kg success rate	0.75kg success rate	1kg success rate
Upright	60%	60%	100%
Parallel to harvester	60%	100%	100%
Perpendicular to harvester	80%	100%	80%
Overall	67%	87%	92%
Overall success rates:			82%

The results from Table 1 show that the robot can pick up weights ranging from 0.5kg to 1kg in all orientations, hence requirements 1.1, 1.2 and 1.3 (the robot shall pick up weights of up to 1kg in weight, the robot shall pick weights in all orientations and the robot shall move the weights) are met.

Weight Storage

Once the weights are picked up by the harvester, they travel down the interior ramp. The ramp is carefully designed with a gradual decline to either side. Weights will be picked up by the harvester and distributed on either side.

A mount was created for the rear servo, so that it can fit snuggly behind the motors. A door to contain real weights is designed with a curved edge, to allow for the rotation of the servo against the edge of the robot. The door is designed to accomplish requirements 1.4 and 1.8, “The robot shall be capable of dropping weights” and “The robot shall be unable to drop a weight if it is considered ‘on board’”. In the doors current state, it achieves requirement 1.8. To complete requirement 1.4, some simple servo code will need to be implemented. A full engineering drawing of the back gate can be found in Appendix C, as well as in Figure 8 which shows that the robot contains all weights that make it ‘on board’ the robot.

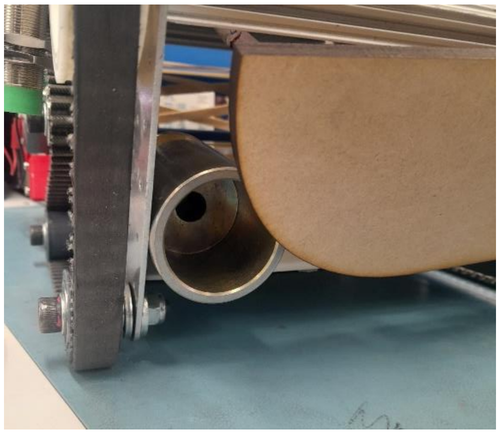
Figure 8 - Back door of robot

Locomotion

For the motion of the robot, tracks were implemented on both sides of the main body. A custom-made drive gear was designed to ensure better grip on the track and an increase in tension. The gear was designed with a flat and grub screw to ensure that there was no slip between the motor and the wheel. Figure 9 displays the final gear design. Track spacers, as seen in Figure 10, were also designed and included over the bearings of the track holders. These were simple parts that increased the radius of the track support bearings. The layout of tracks alongside the drive gear and track spacers can be seen in Figure 11.

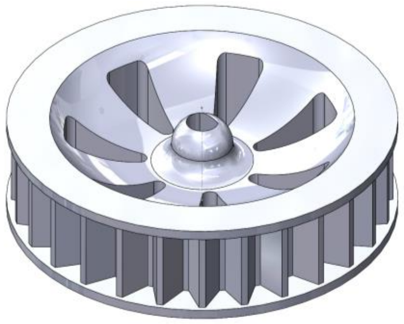
Figure 9 - Drive gear

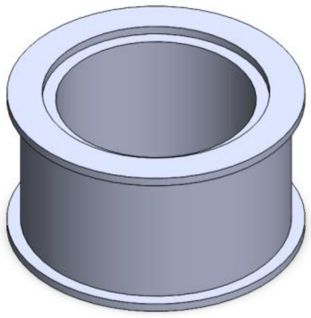
Figure 10 - Track spacers

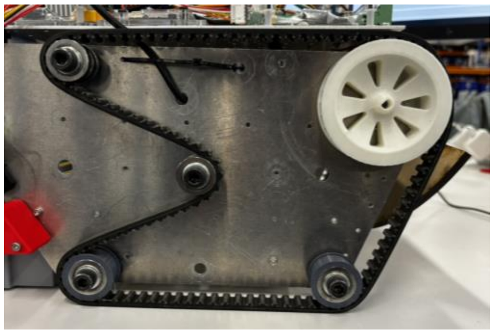
Figure 11 - Drivetrain including belt layout and additional printed parts

To confirm that this design successfully met requirement 1.9 (the robot shall be capable of travelling 4.9m in 20 seconds), the robot was set up with max power forward to both motors. To simulate the proper CPU load, the servo, harvester and sensors were all set up and in a short process to ensure that that the robot would be able to travel this distance. Testing is shown in Table 2, with the final time being 15.67 seconds. This time means that the robot successfully achieves the requirement while still allowing for extra CPU load for other tasks.

Table 2 - Time taken (seconds) to travel 4.9m (arena length)

	Test 1	Test 2	Test 3	Average
Time	15.61	15.96	15.43	15.67

Weight detection

Fully adjustable sensor mounts were manufactured to allow for optimal sensor placement and adjustment. These mounts were designed with 3 degrees of freedom in mind and thus complete control over sensor positioning. This design allows for the sensors to be precisely positioned and angled for optimal weight detection, even after initial assembly and testing. The adjustability of these mounts can be seen in Figure 12.

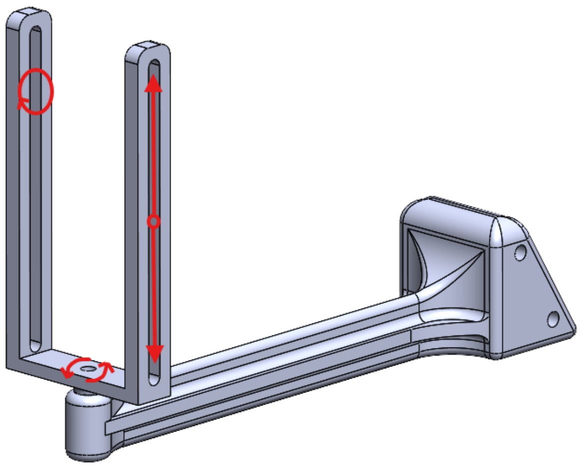

Figure 12 - Sensor Mount with 3DOF

For weight detection, two sets of Time-of-Flight sensors mounted above one another will be purposed for distance sensing. The lower sensor is the shorter ranged sensor (VL53L0X) and the longer ranged sensor (VL53L1X) is above it. These sensors are defined in tof_sensor_init(), further details can be found in Appendix D. Here the sensors are initialized to continuously measure the distance to objects in front of them. This continuous reading is crucial for detecting weights in front of the robot and acting accordingly.

Navigation

Hardware

At the core of the robot is the Teensy 4.0 microcontroller. Used to process data in real time and make decisions based upon programmed logic. Using PWM control, two motors (one left and one right) are signalled through a motor driver circuit to perform accordingly.

As for the navigation sensors used, Time-of-Flight sensors are attached to the robot using the sensor mount in Figure 12. Due to the adjustability of the sensor mount, the position of these sensors can be altered on-the-fly, to further ease testing and development.

Software

The logic for navigation is currently a reactive system, meaning it uses real time data from the ToF sensors to make decisions and react accordingly to the environment around it. This is implemented in the motor_movement() function where the distance values from the sensors are continuously read. The code uses a distance threshold of 150mm. When an obstacle and/or wall are detected within this range, depending on which side, the robot responds accordingly. For example, if an object enters this range on the left, the robot will turn right and vice versa. This ability comes from writing one motor to MAX_FORWARD and the other to MAX_REVERSE, where these variables contain the optimal PWM values for the motor speed. In the case that there is no object detected, both motors are set to MAX_FORWARD, thus allowing for a straight path to be travelled. Full code can be found in Appendix D, with lines 157-167 pertaining to the motor control, which can further be visualized in the finite state machine found in Figure 13.

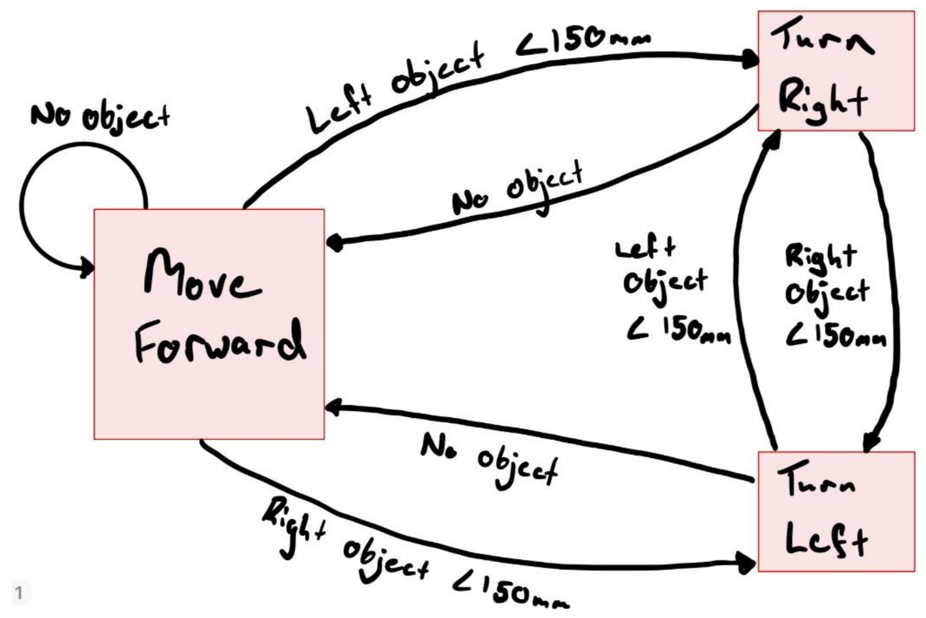

Figure 53 - FSM for reaction-based navigation

Fault Tree Analysis

Discussion

When looking at the Fault Tree Analysis, attached in Appendix E, the top-level event is “Robot Loses Robocup round”. This then branches out into two different cases; “Robot doesn’t collect weights” and “Robot doesn’t collect snitch”. As these are the key scoring metrics, each one of these cases can ultimately result in the robot losing the round due to their respective reasons.

Some branches are more comprehensive than others and thus highlight critical areas that need to be addressed and considered when undergoing further development. Within the “Robot doesn’t collect weights” branch, the biggest faults are movement and/or navigation related. This makes sense, as without functional and reliable movement and navigation, the robot cannot find or move to any weights, thus making it no longer fit for purpose.

In addition to this, the other branches, although smaller, need to be considered and highlight important details that can easily be overlooked. One of which being the weight of the robot, particularly in the case of both robots scoring the same points, and thus the fault of losing due to weight. During development, it is easy to keep adding parts and disregard the weight of the robot, but when looking at previous years, scoring the same points was not an uncommon event and thus this is something that needs to be taken into account for future development for both existing parts and their final iterations, as well as parts that are yet to be developed.

Most parts, if not all, of the robot can fail, showing the effectiveness of doing a Fault Tree Analysis. Therefore, this has been a crucial part of development and will continue to be used for future development. It has highlighted the importance of reliable software and hardware integration, with the biggest takeaway being necessity of having a robot and fault tolerant movement/navigation system. Without this, no other function can be achieved, and the robot fails to meet its intended purpose, to collect weights.

Remaining Tasks

Collection mechanism

Having been the initial main task, very little needs to be done for the collection mechanism from here. The harvester is largely reliable as it has a successful pick up 82% of the time.

Weight storage

For requirement 1.18 (the robot shall have a method of dummy weight detection, or the robot shall have an 80% success rate of dropping the weights at the home base), a diverter might need to be implemented. This will happen if the navigation is unreliable. The diverter implemented would use a servo to divert the dummy weights directly out of the robot, while redirecting the real weights into the storage unit. This is shown in Figure 14.

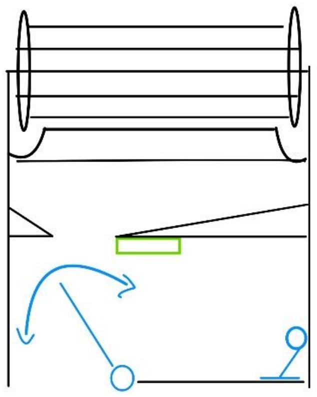

Figure 14 – Weight sorting diverter setup

To achieve requirements 1.14 and 1.15 (the robot should be capable of picking up the snitch and shall have storage for the snitch), a snitch collection bar will be placed inside the robot. As the snitch is higher than the weights, this bar can be placed to slide the snitch directly into the storage as well as stop the snitch from being removed when the weights are dropped at the home base.

Locomotion

Locomotion is mostly complete. The gearing system is effective and throughout testing of other major components did not have errors. The bearing extenders give the robot the extra tension and height desired for the ramp and storage.

A nice to have feature would be the inclusion of the 'mushroom' shaped track protectors as shown in the conceptual report. This will ensure that the tracks do not get caught on the opposition robot, ramps or obstacles, resulting in reliable movement.

Software (navigation and weight detection)

Currently, basic wall detection has been implemented on the robot, using two sets of two ToF sensors. This will need to have an extra sensor mounted above the harvester to determine when there is a wall in front of the harvester. Walls that the robot hit on a corner are currently out of the vision of the ToF sensors. The inclusion of a third direction sensor can cause the robot to avoid getting caught in this position, successfully achieving requirement 1.5 and 1.6 (wall detection in at least three directions and ability to move around the arena, avoiding obstacles, the opposition robot, walls, weights and the snitch).

The implementation of a colour sensor will allow the robot to turn the harvester off if in its own base or the oppositions, hence achieving requirement 1.7.1 and 1.7.2 (The robot should not ‘capture’ any weights from its own base or the opponent’s base). It will also help complete requirement 1.18, in ensuring that the weights are dropped at the correct place.

The navigation algorithm needs to be designed and implemented on the robot. This will use a matrix with a size of 49 by 24, 0.1m x 0.1m squares. Each of these squares will be undetermined until a sensor passes over the area. This will then be given one of three types of possible options, an obstacle, a weight or free space. Knowing the width of the robot, a path can be generated back from the current location and can be executed at the 90 second mark in the event. The robot will follow a wall and make use of the IMU to determine current location and the whereabouts of cleared locations. If dealt with an option to go either way, the robot will turn in the direction where there are more undetermined or free spaces.

A watchdog timer will need to be implemented to ensure that the robot does not end up stuck in a repetitive loop. This will achieve requirement 1.17 (the robot shall have a ‘watchdog timer’ that stops repetitive functions from occurring if they occur for 10 seconds). The timer will be implemented to ensure that if no functions have changed in 10 seconds such as IMU data, sensor readings or track movement, then the code will reset and run a brief reverse of the harvester and tracks to free any stuck weights or tracks.

With the implementation of servos and the colour sensor, code will have to be carefully incorporated to ensure that all elements can function continuously at the same time. To do this, servo code for the back door and internal diversion gate will incorporate a multiple step process where the servo moves small sections until it reaches the desired angle. Although clunky and taking a longer amount of time this should allow the robot to still use the other functions of the harvester and tracks, hence saving time and covering more of the arena.

Requirement changes

Requirements 1.10, 1.11 and 1.12 (the robot shall be capable of turning within a 0.4m diameter circle, the robot shall be able to travel up a gradient of 30 degrees and the robot shall be capable of travelling over a 25mm ‘speed bump’) have been determined to be incorrect.

The gradient of 30 degrees was a misread of 30% which is closer to 17 degrees. This changes requirement 1.11 “The robot should be able to travel up a gradient of 30% (17 degrees)”.

Due to the length the robot requires to have sufficient room to sort and store weights, requirement 1.10 has been changed to “The robot shall be capable of escaping a three walled, 0.4m space”.

Requirement 1.12 has been deemed drastic and reduced to a height of 15mm as the previous years (all with the same requirement) have never had a bumper higher than 15mm (the requirement to get onto the ramp). Thus, requirement 1.12 has been changed to “The robot shall be capable of travelling over a 15mm ‘speed bump’”.

Conclusion

The current robot meets most of the core requirements or are undergoing later stages of development. The combine harvester style collection mechanism has had significant design changes and iterations. Now achieving an 82% overall pick up success rate across the most likely weight orientations and sizes. Locomotion testing confirms the speed requirement is comfortably met, with margin for additional load, while the track and gearing provides reliable and controlled movement. The weight storage is mechanically functional, though dependent on further development of the rear gate to meet the drop-off requirement. Navigation remains in early development, currently using a basic wall-avoidance algorithm to avoid colliding with any obstacles. Though it does not currently have any ability to determine its own location, nor the location of the home base for drop off. And so higher-level path planning is yet to be addressed. These elements, along with the addition of a centered sensor, and the implementation of a watchdog protocol, represent the main factors left for the goals for navigation to be met.

Appendices

Appendix A: Engineering drawing of harvester disk (Find attached)

Appendix B: Time taken to pickup a weight

0.5kg	Upright	Parallel (to harvester)	Perpendicular (to harvester)
Left	Fail	4.58	6.36
Middle	4.77, 4.25, 19.33	Fail, 3.74, Fail	4.84, Fail, 19.13
Right	Fail	4.23	14.97
Success rate	60%	60%	80%
Overall, 0.5kg success rate:			67%
0.75kg	Upright	Parallel (to harvester)	Perpendicular (to harvester)
Left	Fail	4.00	14.18
Middle	5.32, 6.22, 19.45	6.61, 5.43, 13.33	5.70, 6.03, 6.21
Right	Fail	3.21	7.03
Success rate	60%	100%	100%
Overall, 0.75kg success rate:			87%
1kg	Upright	Parallel (to harvester)	Perpendicular (to harvester)
Left	6.87	5.12	4.78
Middle	7, 6.35, 5.04	10.93, 6.93, 13.27	5.75, Fail, 9.02
Right	6.60	5.30	6.15
Success rate	100%	100%	80%
Overall, 1kg success rate:			93%
Overall success rate:			82%

Appendix C: Engineering drawing of back gate (Find attached)

Appendix D: Main code

```
#include <Servo.h> // Include the Servo library for pulse control
#include <Wire.h>
#include <VL53L0X.h>
#include <VL53L1X.h>
#include <SparkFunSX1509.h>


// DRIVE MOTOR INITIALISERS
// Setup for motors using 2 PPM Motor driver modules, plugged into CON65 and CON66
// CON65 powers the harvester
// CON66 powers the drive motors
int MAX_REVERSE = 1050;
int MAX_FORWARD = 1950;

// Define motor pins
int MotorPinR = 7; // PWM-capable pin for motor left
int MotorPinL = 8; // PWM-capable pin for motor right
int DC_MOTOR = 0; // PWM-capable pin for combine

// Create Servo objects to control the motors
Servo motorL;
Servo motorR;
Servo dc_motor;
```

```
// TOF SENSOR SETUP
// Setup for the tofs, L0 plugged into CON28 and CON29, L1 plugged into CON32 and CON33
// Requires IC plugs CON60 to CON26, CON61 to CON35, CON52 to CON9
const byte SX1509_ADDRESS = 0x3F;
#define VL53L0X_ADDRESS_START 0x30
#define VL53L1X_ADDRESS_START 0x35

// The number of sensors in your system.
const uint8_t sensorCount = 2;

// The Arduino pin connected to the XSHUT pin of each sensor.
const uint8_t xshutPinsL0[8] = {1,6};
const uint8_t xshutPinsL1[8] = {2,5};

SX1509 io; // Create an SX1509 object to be used throughout
VL53L0X sensorsL0[sensorCount];
VL53L1X sensorsL1[sensorCount];

// INDUCTIVE SENSOR SETUP
// 8 pin Digital pin used in CON55 to Inductive Level shift, with sensor set off CON5
int inductive_sensor_pin = 2;
int inductive_sensor = 0;


// Servo Setup
// Plugged into the digital shift board, using CON54
Servo servoDoor; // create servo object to control a servo
Servo servoGate;
int servoDoorPin = 33;
int servoGatePin = 32;

void tof_sensor_init(void) {
Serial.begin(9600);

while(!io.begin(SX1509_ADDRESS)){
Serial.print("Failed to talk to io expander\n");
delay(1000);
}

Wire.begin();
Wire.setClock(400000); // use 400 kHz I2C

// Disable/reset all sensors by driving their XSHUT pins low.
for (uint8_t i = 0; i < sensorCount; i++)
{
io.pinMode(xshutPinsL0[i], OUTPUT);
io.digitalWrite(xshutPinsL0[i], LOW);
io.pinMode(xshutPinsL1[i], OUTPUT);
io.digitalWrite(xshutPinsL1[i], LOW);
}

// L0 Enable, initialize, and start each sensor, one by one.
for (uint8_t i = 0; i < sensorCount; i++)
{
// Stop driving this sensor's XSHUT low. This should allow the carrier
// board to pull it high. (We do NOT want to drive XSHUT high since it is
// not level shifted.) Then wait a bit for the sensor to start up.
io.pinMode(xshutPinsL0[i], INPUT);
delay(10);

sensorsL0[i].setTimeout(500);
if (!sensorsL0[i].init())
{
Serial.print("Failed to detect and initialize sensor L0 ");
Serial.println(i);
while (1);
}
```

```
// Each sensor must have its address changed to a unique value other than
// the default of 0x29 (except for the last one, which could be left at
// the default). To make it simple, we'll just count up from 0x2A.
sensorsL0[i].setAddress(VL53L0X_ADDRESS_START + i);

sensorsL0[i].startContinuous(50);
}

// L1 Enable, initialize, and start each sensor, one by one.
for (uint8_t i = 0; i < sensorCount; i++)
{
// Stop driving this sensor's XSHUT low. This should allow the carrier
// board to pull it high. (We do NOT want to drive XSHUT high since it is
// not level shifted.) Then wait a bit for the sensor to start up.
io.pinMode(xshutPinsL1[i], INPUT);
delay(10);

sensorsL1[i].setTimeout(500);
if (!sensorsL1[i].init())
{
Serial.print("Failed to detect and initialize sensor L1 ");
Serial.println(i);
while (1);
}

// Each sensor must have its address changed to a unique value other than
// the default of 0x29 (except for the last one, which could be left at
// the default). To make it simple, we'll just count up from 0x2A.
sensorsL1[i].setAddress(VL53L1X_ADDRESS_START + i);

sensorsL1[i].startContinuous(50);
}
}

void inductive_sensor_init()
{
pinMode(inductive_sensor_pin, INPUT);
Serial.println("Setup COMPLETE");
}


void setup() {
// DRIVE MOTOR INITIALISERS
// Attach the motor control pins to the Servo objects
motorL.attach(MotorPinL);
motorR.attach(MotorPinR);
dc_motor.attach(DC_MOTOR);

servoDoor.attach(servoDoorPin);
servoGate.attach(servoGatePin);

// TOF SENSOR SETUP
tof_sensor_init();

// INDUCTIVE SENSOR SETUP
inductive_sensor_init();

Serial.print('\t');
Serial.print("SETUP COMPLETE"); // DO NOT DELETE
Serial.print('\t'); // DO NOT DELETE
}

void motor_movement() {
if ((sensorsL0[0].readRangeContinuousMillimeters() < 150) && (sensorsL1[0].readRangeContinuousMillimeters() < 150)) {
motorL.writeMicroseconds(MAX_REVERSE);
motorR.writeMicroseconds(MAX_FORWARD);
} else if ((sensorsL0[1].readRangeContinuousMillimeters() < 150) && (sensorsL1[1].readRangeContinuousMillimeters() < 150)) {
motorL.writeMicroseconds(MAX_FORWARD);
motorR.writeMicroseconds(MAX_REVERSE);
```

```
} else {
motorL.writeMicroseconds(MAX_FORWARD);
motorR.writeMicroseconds(MAX_FORWARD);
}
}

void tof_sensor_read() {
for (uint8_t i = 0; i < sensorCount; i++)
{
Serial.print(sensorsL0[i].readRangeContinuousMillimeters());
if (sensorsL0[i].timeoutOccurred()) { Serial.print(" TIMEOUT"); }
Serial.print('\t');
}

for (uint8_t i = 0; i < sensorCount; i++)
{
Serial.print(sensorsL1[i].read());
if (sensorsL1[i].timeoutOccurred()) { Serial.print(" TIMEOUT"); }
Serial.print('\t');
}
Serial.println();
}

void inductive_sensor_read() {
inductive_sensor = digitalRead(inductive_sensor_pin);
Serial.println(inductive_sensor);
}

void servo_door() {
servoDoor.writeMicroseconds(1000); // sets the servo position full speed backward
delay(100); // waits for the servo to get there
servoDoor.writeMicroseconds(2000); // sets the servo position full speed forward
delay(100); // waits for the servo to get there
}

void servo_gate() {
servoGate.writeMicroseconds(1000); // sets the servo position full speed backward
delay(100); // waits for the servo to get there
servoGate.writeMicroseconds(2000); // sets the servo position full speed forward
delay(100); // waits for the servo to get there
}

void loop() {
//Set direction for all channels
// DRIVE MOTOR LOOP CODE

motor_movement();
dc_motor.writeMicroseconds(MAX_FORWARD);

tof_sensor_read();

inductive_sensor_read();

servo_door(); // Takes time to run
servo_gate();
}
```

Appendix E: Fault Tree Analysis (Find attached)