ELIZABETH'S ORCHARD

ROS SIMULATION

USER MANUAL

EORC065
CCHA504
FELZ123
ZALL 747
CPAR831
CEIR349
GTAK640
CBRO691

INTRODUCTION

R.O.S. (Robotic Orchard Simulator) is a simulation of harvesting robots in a kiwifruit orchard. The simulation models different types of harvesting robots, people, animals, tractors and the environment of a kiwifruit orchard.

Each dynamic object within the simulation emulates the various behaviours that is expected from the real world variables the objects model after. This is then supported by basic configuration of variables available to the user to simulate various scenarios.

The behaviours that the simulation presents are in turn prepared for these various scenarios, as the ultimate goal of picking kiwifruit at an optimal rate is pursued. The scenarios available to simulate include those expected and unexpected, as each dynamic object adjusts itself accordingly to the possible situations that may present itself.

Additionally, the simulation implements individual robots that pass messages between each other, in response to the simulation of the environment. This in turn implements interactions between robots and the environment. Test cases have also been implemented to demonstrate these features.

ASSUMPTIONS / LIMITATIONS

To ensure a smoothly running simulation, several assumptions had to be made and limitations to the simulations were applied. This first lead to the design decision where the kiwifruit was not to be modelled, neither on the trees or whilst being picked. This was due to the understanding that it would add unnecessary complication to the simulation.

This then led to the second design decision of having one row equal to a full bin of kiwifruit. This simplified the process of kiwifruit picking and allows for the simulation to better show the behaviours of the elements modelled.

The roofed struaActure of the kiwifruit orchard is only partially modelled. This is to show the basic idea of what the kiwifruit orchard looks like while still having high visibility of the simulation.

Visually, the model of the robots are shown as only boxes. This was done to accurately show the dimensions of the robots as specified. Each robot is coloured differently so they are easily differentiable from both the environment and each other.

The bins have been programmed to follow behind the robot they are currently being carried by. It is assumed that there is a connector between the robot and the bin. The connector is not modelled in the simulation to maintain the simplicity and easeof understanding.

Although this simulation offers configuration there are limits to the maximum values. The number of kiwifruit rows is limited to 20. This is to ensure that the stage does not become too large and is easily viewable. Similarly the pickers have a maximum value that is equal to the number of rows. This is so that there are no idle pickers. The number of carriers is automatically generated based on the number of pickers. There are 7 carriers to every 10 pickers. This ratio allows for efficiency and represents a realistic situation.

ELEMENTS OF STAGE

This stage within the simulation includes both static and dynamic elements. The stage has a 1:1 scale where 1 unit on the stage is equal to 1 metre.

STATIC ELEMENTS

The Trees:
The kiwifruit trees are 1.8m tall and create a partial roof of vines across the rows.
Halfway between each tree is a supporting post.
Weeds:
Tall weeds tend to grow on orchards creating obstacles to be navigated. The weeds are 0.5 x 0.5 x 0.5

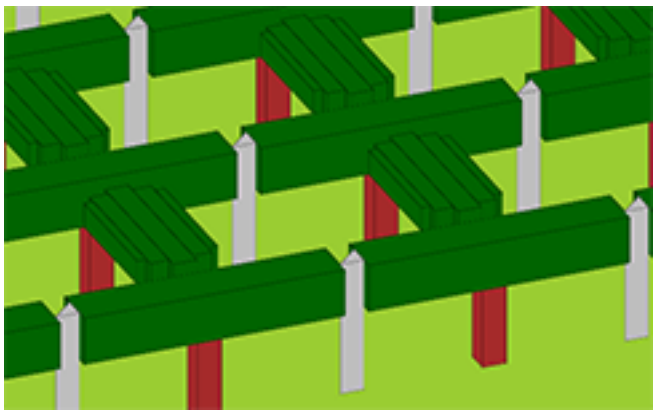


Figure 1.) Tree structures

Figure 2.) Weed

DYNAMIC ELEMENTS

Animals:
Farm pets such as cats and dogs tend to wander into the orchard boundries getting in the way of workers and robots alike.
Their dimensions are 1.0 x 0.8 x 0.7

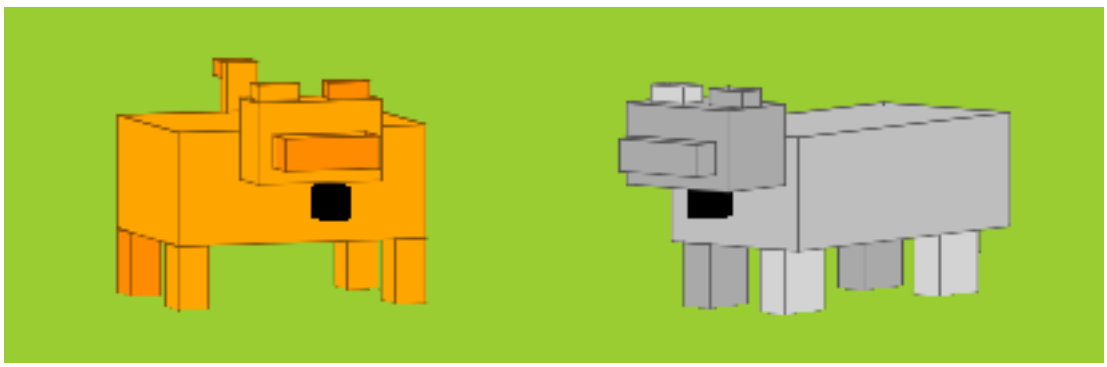


Figure 3.) A dog and a cat

People:
Workers on the orchard patrol the area ensuring everything is working correctly. Visitors also come into the orchard and wander about looking around the grounds.
Their dimensions are 0.55 x 1.0 x 1.7

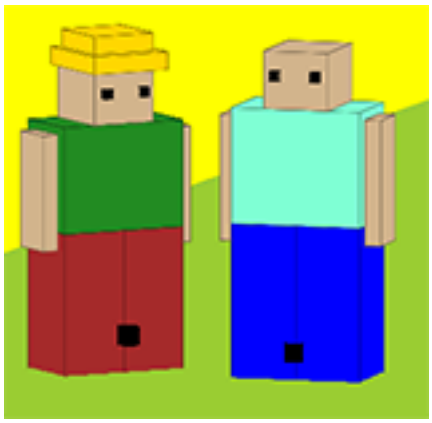


Figure 4.) A worker and a visitor


Figure 5.) A tractor with people

Robots:
The picker and the carrier are the harvesting robots performing tasks on the orchard. The picker stays on the rows picking the kiwifruit from the trees while the carrier transports the full bins of kiwifruit from the picker to the collection station.
The picker (pink) is 2.5 x 1.5 x 1.0
The carrier (blue) is 1.8 x 1.5 x 0.5

Bins:
The bins hold the kiwifruit that is picked.
They are transported by either a picker or a carrier.
The bins are 1.0 x 1.0 x 1.0

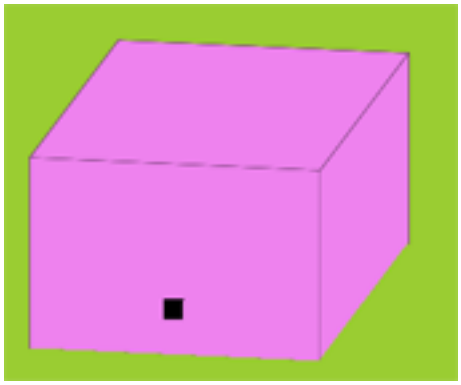


Figure 6.) A picker

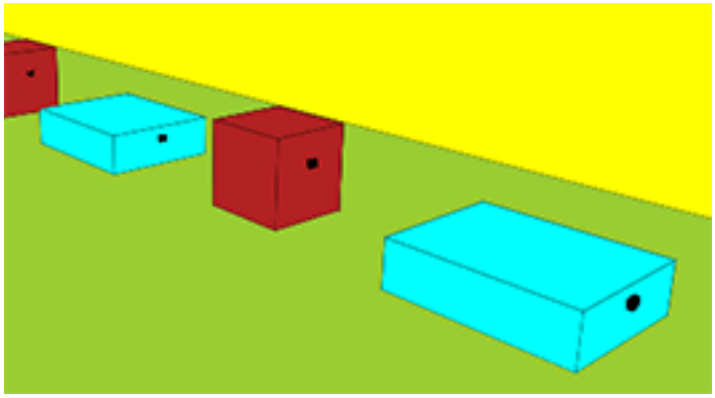


Figure 7.) The queue of carriers

SETTING UP

To set the project up, there are several steps required.

1.) ROS Indigo must be installed
2.) The project repository must be cloned into the user's /Home directory
3.) The various configurations available to the user should be set up to suit the user's desired needs.

ROS Installation

ROS Indigo must be installed on your linux machine. If ROS Indigo is not yet installed, please follow the steps outlined here:
http://wiki.ros.org/ROS/Installation

Cloning the Repository

1) Download the zip file from github and unzip in your /Home directory.
   OR type '**git clone https://github.com/Lavitasy/threeohsix.git**' into your terminal after navigating to your /Home directory.
2) Follow the instructions that Github gives through the terminal to complete cloning the repository.

Configuring the World

1) Before starting the simulation the user can configure some of the orchard settings (number of rows, number of picker robos, and the width of rows) by typing '**python GUI.py**' in the /threeohsix direcotyr.
A window will then pop up prompting the user to enter their own values.

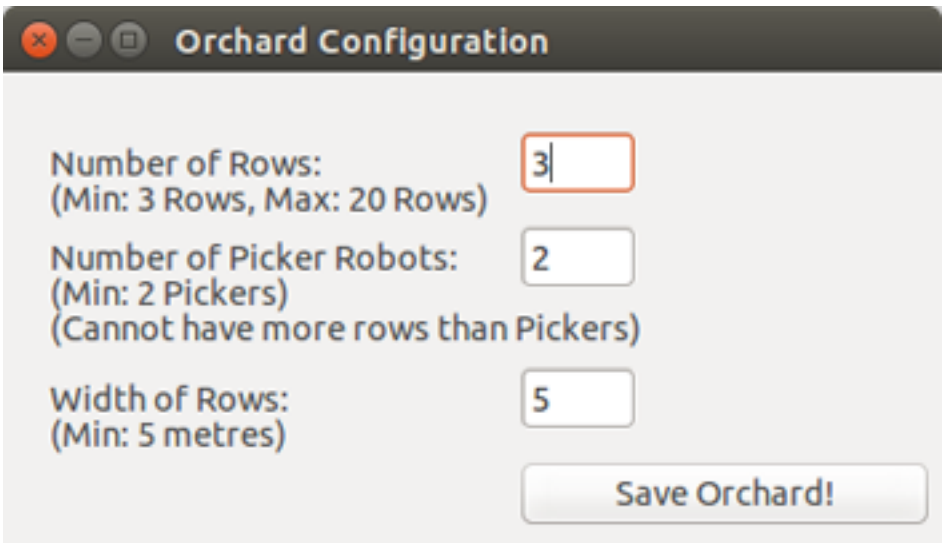


Figure 8.) The configuration GUI

Starting the Simulation

4) Type '**bash run.sh'** or '**bash run_one_window.sh**' from the /threeohsix directory, and the simulation will begin!

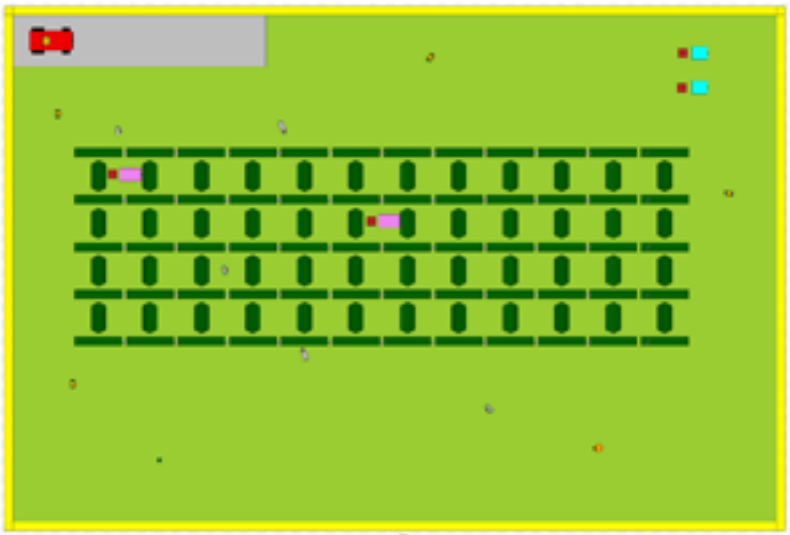


Figure 9.) The world on startup

BEHAVIOURS

SENSING BEHAVIOUR

Aim:

Ensure the robots can use sensors to sense their environment and react appropriately.

Use:

In every dynamic object

Implementation:

- The laser scanner retrieves data regarding the environment
- Node is analysing data received to decide the obstacle's direction
- Node turns to avoid colliding with the obstacle

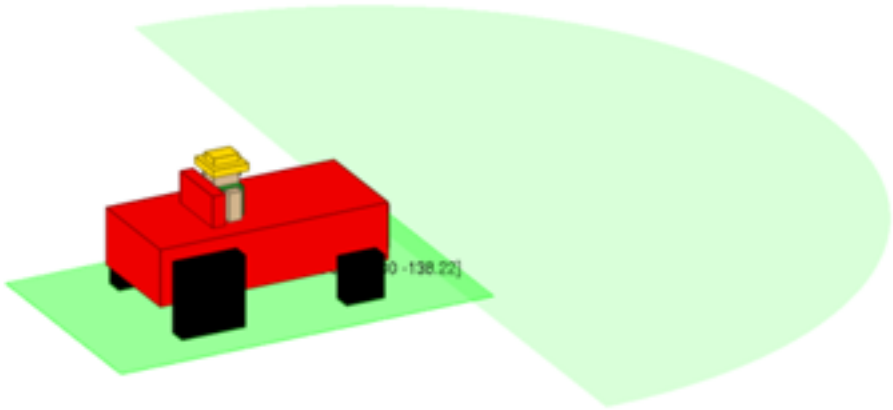
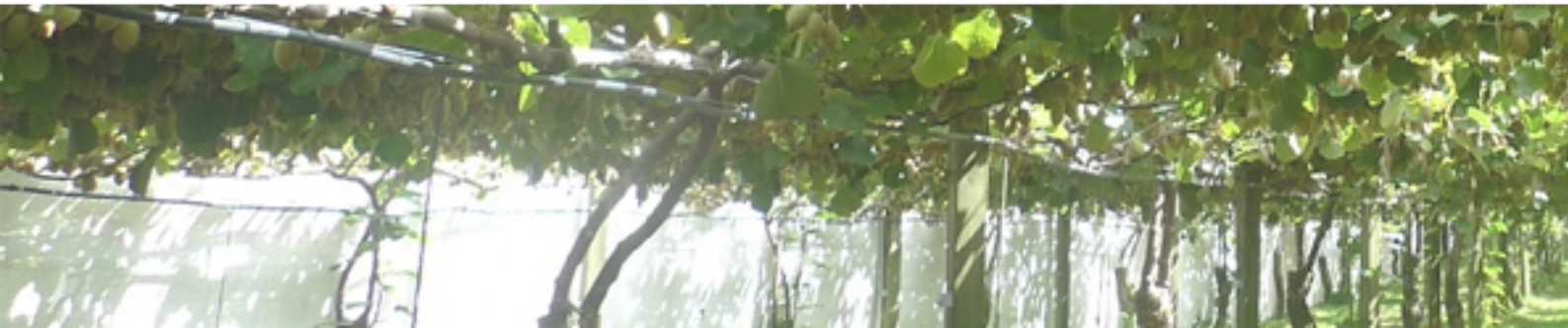


Figure 10.) A tractor with its range of sense



TURNING BEHAVIOUR

Aim:

For a robot to have the ability to turn exactly 90 degrees clockwise and and anticlockwise

Use:

Every dynamic object, except animals, as their behaviour in terms of where they go can be considered random

Implementation:

- The method modifies the twist.angular.z value.
- It is understood that twist.angular.z is in terms of angular velocity so this is set to pi/8, meaning the robot will rotate pi/8 radians per second (one full 360 degree rotation every 16 seconds.
- The rospy is set to sleep for 4 seconds, after which twist is initialised again.
- This initialisation resets the values of twist, stopping the robot
- After this 4 second sleep, the robot should have rotated 90 degrees (pi/2 radians).

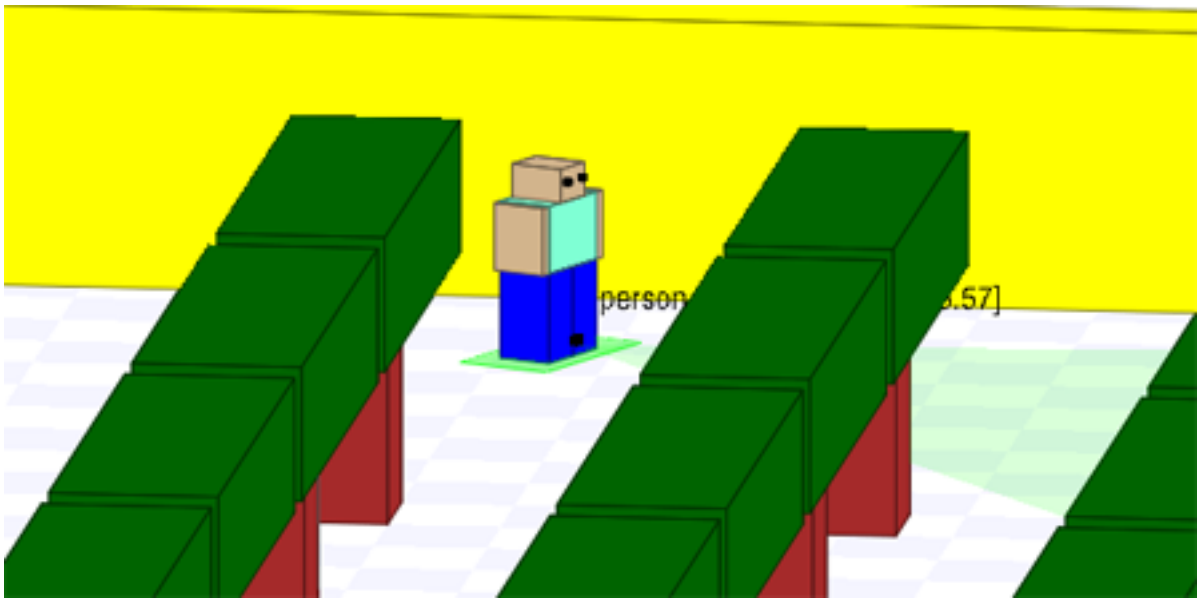
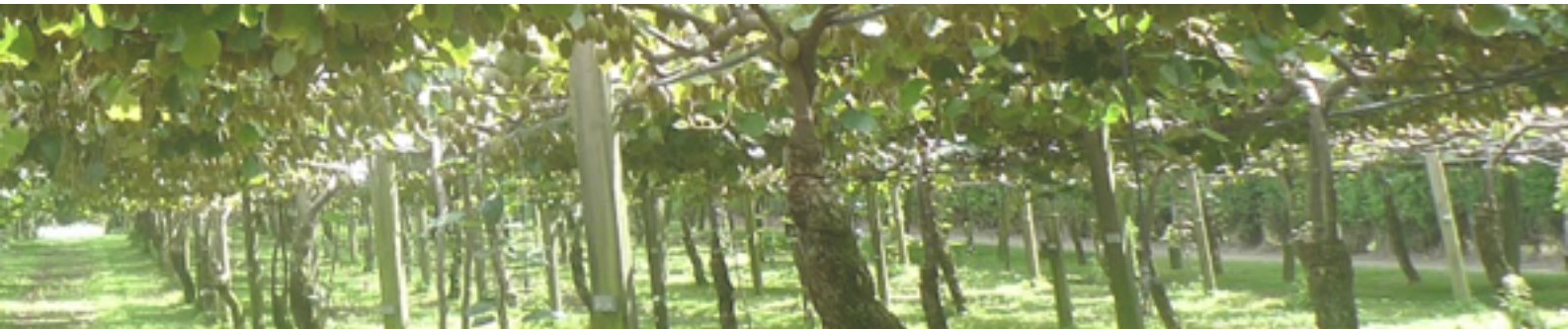


Figure 11.) A person mid-turn



DIRECTED MOVEMENT

Aim:

Moves robot to a destination

Use:

In every dynamic object

Implementation:

- Node can move x metres forward
  - Uses the amount of time to move x metres by the node's velocity rounded to the nearest second
  - Publishes its velocity for this amount of time, then if needed moves at a slower velocity to move a total x steps and then stops
- Node can move to a position on the map given by a point
  - The robot retrieves data regarding its position in the world
  - Utilises turning and move x metres functionality to get to the new position
  - Firstly moves in the x or y direction depending on its current face direction in the world (North, South, East, West)
- Both methods co-operate with sensors to ensure if meet with obstacles that it'd still end at the new position

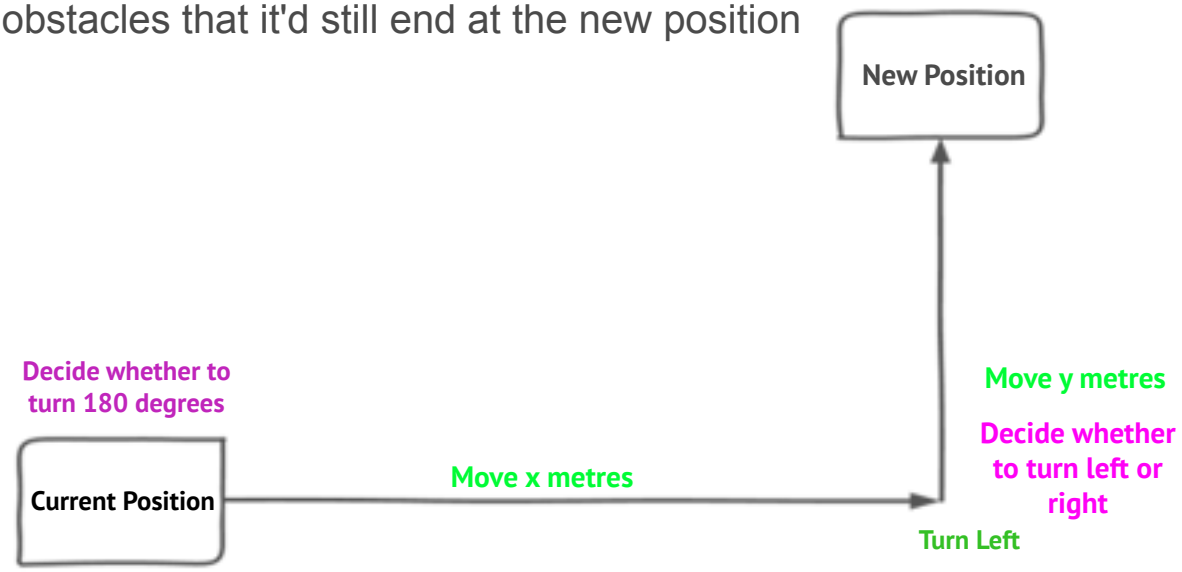


Figure 12.) Procedure of move

AWARENESS

Aim:

Make robots able to be aware of their current positions, and adjust themselves if there is an error

Use:

All dynamic objects that do not move randomly

Implementation:

- Robots know their current orientation as a euler value, converted from quaternion values
- Robots also know approximately which face they are facing (NORTH SOUTH EAST WEST)
- Robots then calculate the distance in radians of their current orientation to the nearest face
- If they are facing a face perfectly, do nothing
- If not, they rotate towards the nearest face until they are exactly facing a face.

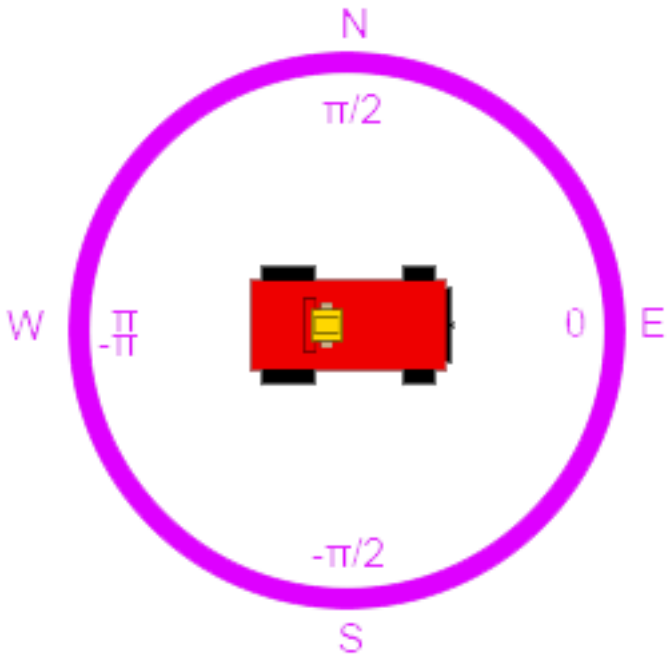


Figure 13.) A robot's range of orientation

FOLLOWING BEHAVIOUR

Aim:

Make one robot follow the other

Use:

Bins following their associated robot to simulate carrying of the bin.

Implementation:

- Harvesting-robot publishes its velocity to a rostopic, which is received by a subscribing bin.
- Bin extracts the velocity from the message received, and sets its own velocity accordingly.
- Standard messages are sent to categorize following behavior. For example, full bins send a request to be picked to carrier robots.

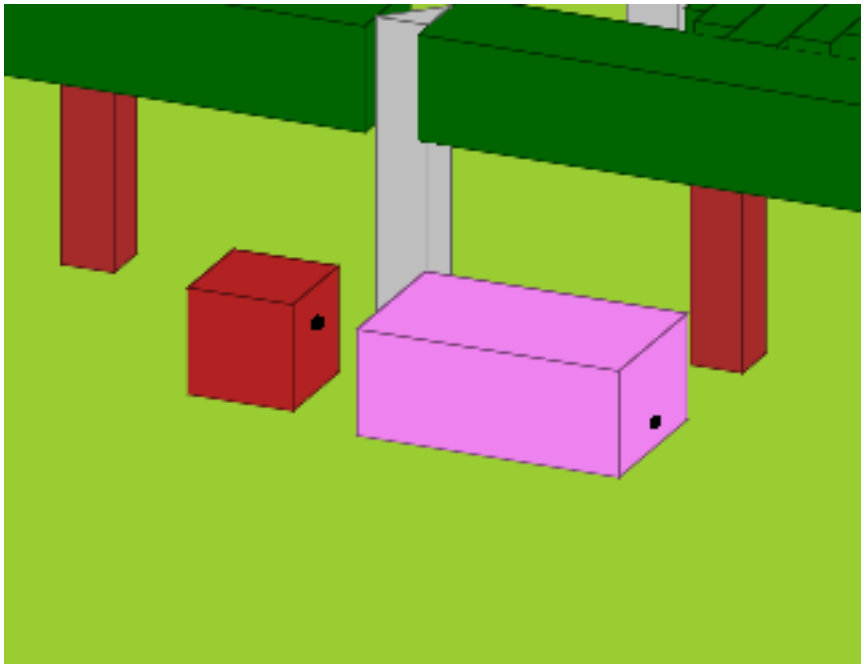


Figure 14.) A bin following a picker

RANDOM ANIMALS

Aim:

To make an animal class move around the world randomly.

Use:

Adds a dynamic obstacle that the robots have to avoid. Less structured than the other classes so useful for testing unexpected interactions.

Implementation:

The animal randomly changes it's speed or orientation every 3 seconds

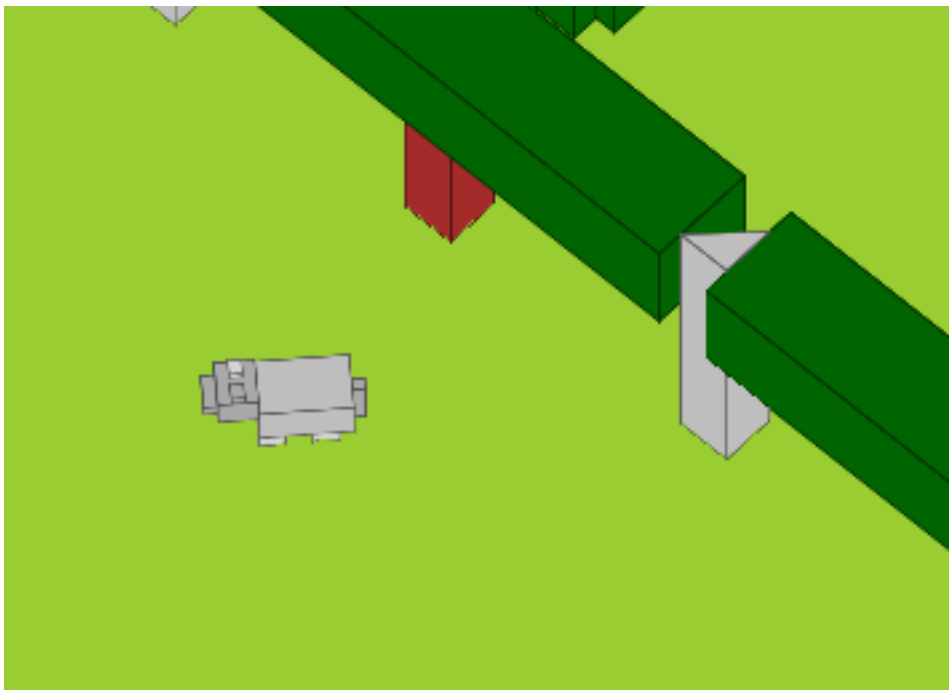


Figure 15.) An animal moving around

ROBOT OVERVIEW

PICKERS

The picker robots travel up and down the lanes of kiwifruit collecting the fruit. They pull a bin and fill it with the fruit. When the bin is full the picker will wait for a carrier to remove the full bin and replace it with an empty one. Once the picker has an empty bin it will continue down the next kiwifruit lane to continue harvesting.

GENERAL BEHAVIOURS

Pickers begin the simulation at the head of the lane they will be harvesting the kiwifruit from. The key features in the implementation of pickers are:

Obstacle detection : When an obstacle is detected the picker will stop and wait until the obstacle has moved.
Attaching bins : The pickers are able to attach bins to themselves to transport them through the kiwifruit lanes
Detaching bin : The pickers are able to detach the bins once they are full
Moving through the lanes : The pickers are able to navigate up and down through multiple kiwifruit lanes.
Call for a carrier : The pickers are able to call for a carrier to swap their full bin for an empty one

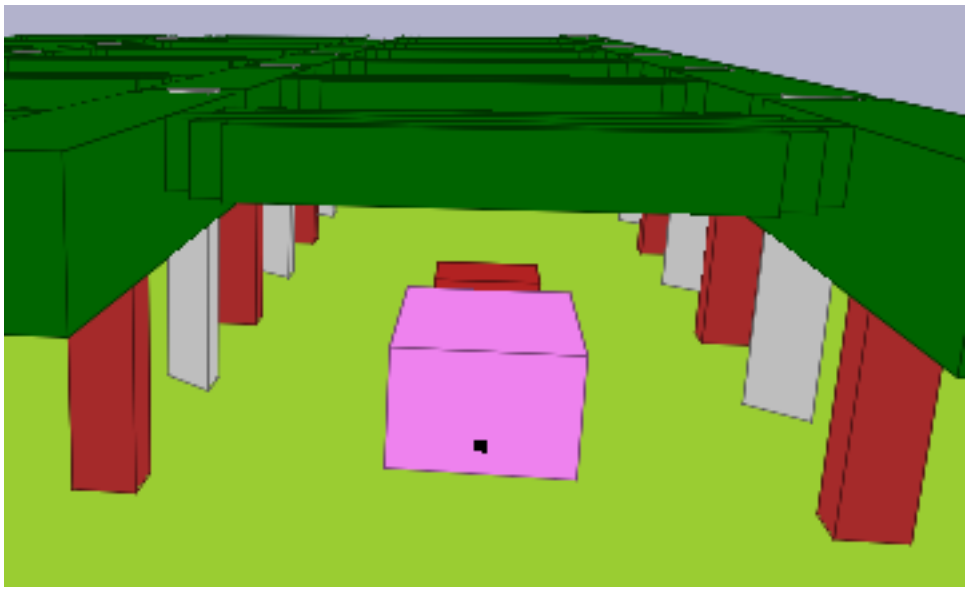


Figure 16.) A picker going through the orchard

CARRIERS

The carrier robots were designed to assist in the movement of the bins. The pickers require new bins at the end of each row, as the bins they are currently holding are full at that point, and so the carriers are required to assist in that, by bringing the new bin to the picker, and removing the full one.

GENERAL BEHAVIOURS

Carriers are originally lined up in a queue at the top right corner of the orchard waiting to be called. Once they are called, they change their state, to stop receiving calls while at work. The key features in the implementation of the carrier robots are:

Obstacle detection : When an obstacle is detected the carrier will stop and wait until the obstacle has moved. If the carrier is in range of the bin it wants to pick up, it will temporarily turn off it's detection.
Attaching bins : The carriers are able to attach bins to themselves to assist in transporting them around the orchard
Detaching bin : The carriers are able to detach the bins once they have reached their final destination
Moving around the orchard : The carriers are able to navigate the orchard
Reacting to the bin calls : The carriers respond to bins calling to be picked up. It uses the detach and attach bin methods to swap with a picker an empty bin with a full one. After that, it carries the full bin all the way around the orchard until the tractor driveway to load off the fruits. Consequently, it carries the now empty bin back to the waiting queue, expecting to be called again later.
Reacting to the pickers calls : The carriers respond when the pickers call for swapping a full bin with an new empty bin.

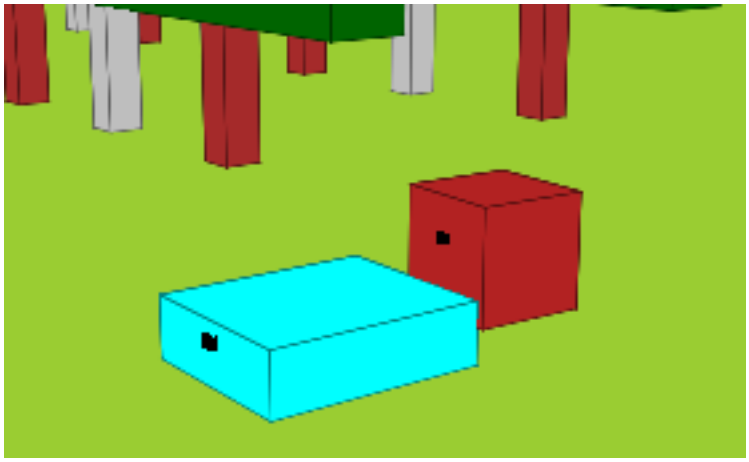


Figure 17.) A carrier and a bin

BINS

In the early stages of our design it was decided upon that the bins would be dynamic themselves, as to better create a simulation of the world. The bins "attach" and "detach" from picker and carrier robots by listening to their speed and angles.

GENERAL BEHAVIOURS

The key features in the implementation of the carrier robots are;

Attaching : The bins are able to attach themselves to a given robot and follow them around the orchard
Detaching : The bins allow for themselves to detach from the robot they are currently connected to.
Communication with other robots : The bins are able to communicate whether they are full or empty to other robots

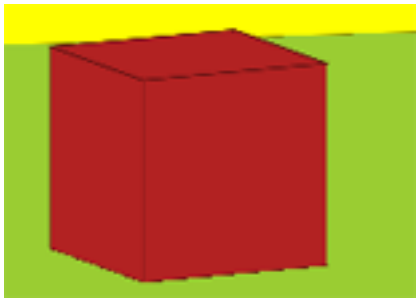


Figure 18.) A bin

WEEDS

Weeds are randomly placed throughout the orchard.

GENERAL BEHAVIOURS

Weeds begin the simulation in random positions along the top and the bottom of the orchard. The key features in the implementation of the weeds are;

Detection : When the weed detects a moving object within a metre of itself it will simulate being run over and flattened by disappearing

Figure 19.) Weed

PEOPLE

There are two types of people found within the orchard, workers and visitors. The people in the world serve as obstacles for the other dynamic objects in the simulation.

GENERAL BEHAVIOUR

Workers and visitors begin the simulation standing around the edge of the orchard. The key features in the implementation of pickers are:

Obstacle detection : When an obstacle is detected a person will turn to avoid the obstacle and will continue moving in the new direction.
Movement : People move in a straight line, only turning when there is an obstacle in their path.

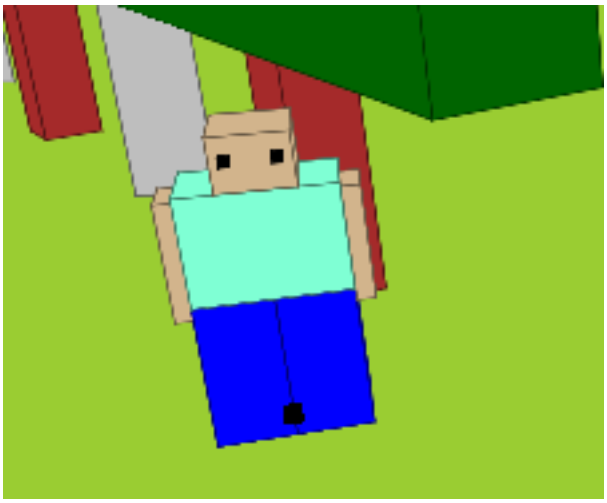


Figure 20.) A visitor

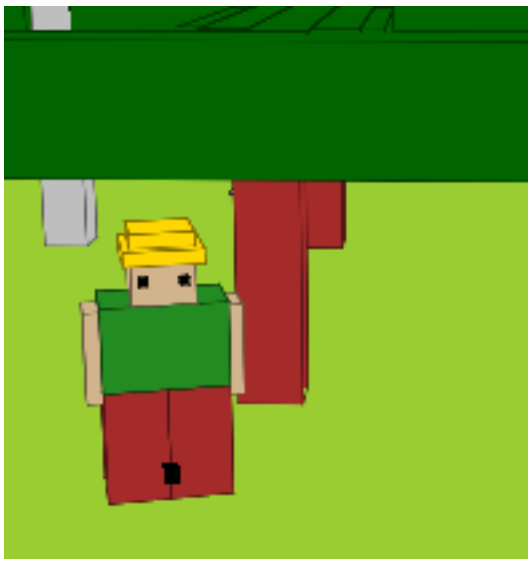


Figure 21.) A worker

TRACTORS

The tractor stays on the driveway throughout the simulation. It will wait at each end of the driveway simulating work such as being loaded with kiwifruit bins or being unloaded of empty bins.

GENERAL BEHAVIOURS

The tractor begins the simulation at the top left of the screen on the driveway. The key features in the implementation of the tractor are;

Obstacle detection : The tractor stops when it detects an obstacle and will wait for it to move simulating the real world safety factors of a tractor.
Movement : The tractor drives up and down the driveway. When it reaches one end of the driveway it will turn 180 and continue.

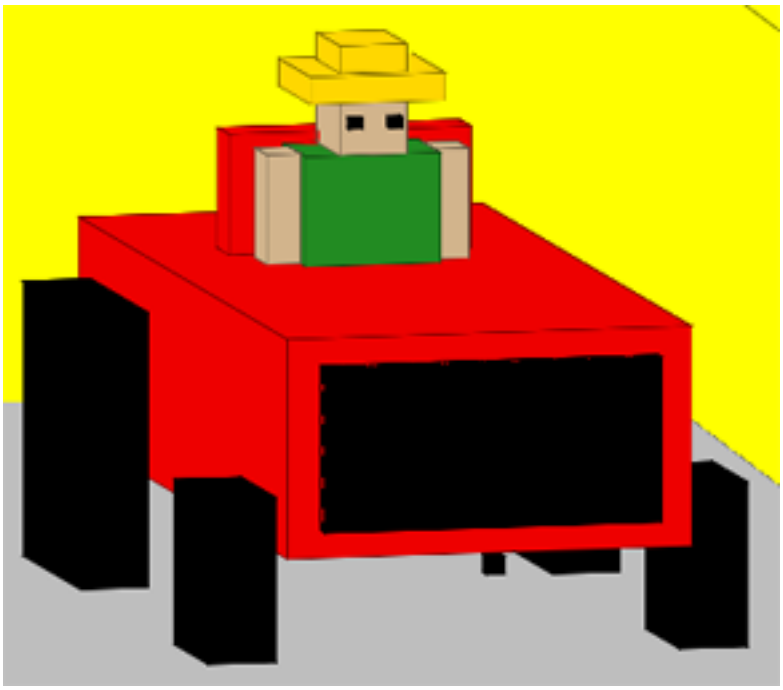


Figure 22.) A tractor

ANIMALS

There are two types of animals found within the orchard, cats and dogs. They randomly move about the orchard creating obstacles.

GENERAL BEHAVIOURS

Cats and dogs begin the simulation stood in the orchard. The key features in the implementation of the animals are;

Obstacle detection : When a animal detects an obstacle in its path it will turn to avoid the obstacle and continue moving in the new direction.
Random movement : Animals will randomly change their direction and/or their speed.

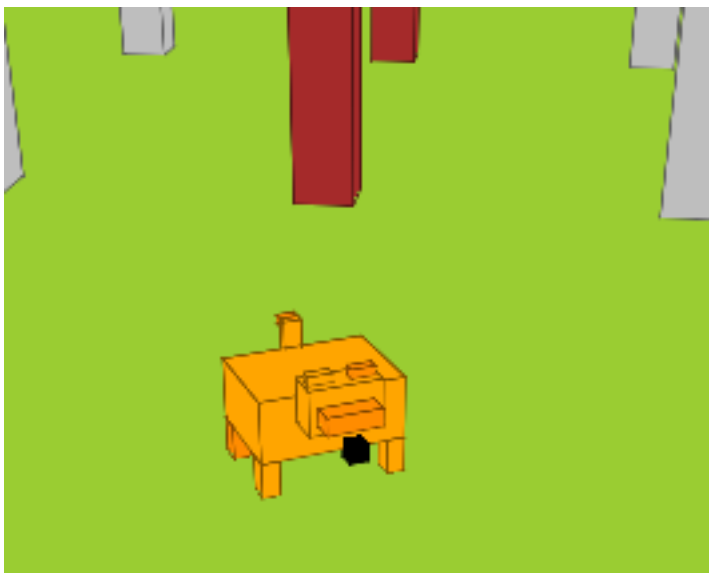


Figure 23.) A dog

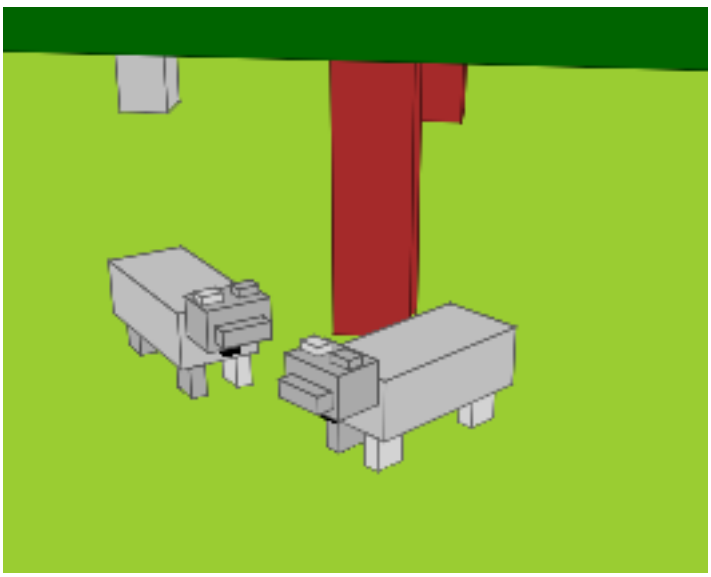


Figure 24.) A pair of cats

TESTING

To test the project, please use the following steps:

1.) Ensure the terminal is in the project directory /Home/threeohsix
2.) Run **'roscore'** in the terminal
3.) Run **'catkin_make'** in the terminal

PYTHON TEST

To run python tests:

1.) Run '**source devel/setup.bash**' in the terminal
2.) Run '**robotsim test_node.py**' in the terminal

ROSTEST

To run rostest

1.) Run '**source devel/setup.bash**' in the terminal
2.) Run '**rosrun robotsim weed.py 11**' in the terminal

Then open a new tab in the terminal

3.) Then run '**source devel/setup.bash**' in the new tab of the terminal
4.) Run '**rosrun stage_ros stageros src/robotsim/world/orchard.world**' in the terminal

Then open a new tab in the terminal

5.) Then run '**source devel/setup.bash**' in the new tab of the terminal
6.) Run '**cd src/robotsim/scripts**' in the terminal
7.) Run '**./test_movement.py**' in the terminal