IDP Documentation - Software

Team: root g
Robots: e, π

IDP Group L102

February 22, 12021

Downing College

Name	Lab Group
Cameron Spiers	120
Haymandhra Pillai	120
Akash Gupta	124
Noah Gordon	124
Pengyu Zhang	125
Tommy Rochuseen	126

Assessor	Prof Tim W.
Advisor	Caglar K.

Link to Repository: https://github.com/Guppy16/idp-L102

1 Software Approach

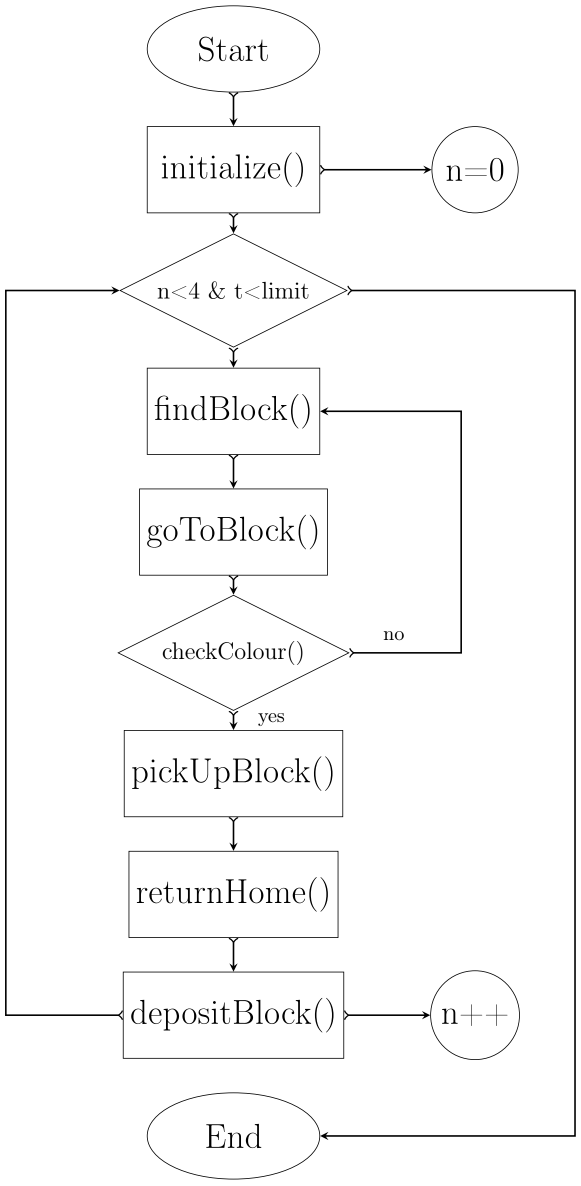

Figure 1: Flow chart showing the top level functions used in one control loop. The timer is set to run for 2.5 minutes, so that each robot can take it in turn to find a block

2 Structure

- Only one controller was used
- Note that one loop in the flowchart doesn't correspond to one timestep, but the loop breaks after 2 minutes to allow the robot to return home
- A file was kept to store the initial position of the robots and threshold values for the colour sensor
- Our approach was to split the code into 3 main modules: Robocar class, Controller module, Utility functions

2.1 Robocar Class

The robocar class extends the robot class provided in Webots. Robocar is designed to initialises the sensors and flags, and provides handy functions for sensing and movement.

- init - initialise flags, sensors and actuators
- Sensor functions: update_sensors, getHeadingDegrees, get_ds_sensor_object_pos, detect_block_colour
- Driving functions: go_forward, go_backwards, turn_left, turn_right, turn_and_drive, stop, rotate
- Integrated functions: rotate_to_bearing, return_home, get_other_robot_pos

2.2 Controller Module

- find_blocks
- rotate_to_target_block
- check_block_colour
- drive_around_block
- check_front_clear
- go

2.3 Utility Functions

- next_block - finds the closest block in a list, which hasn't been picked up
- ds_sensor_to_m - converts the distance sensor value to a distance using linear interpolation from a lookup table
- is_within_range - checks if two position vectors are close to each other
- getLocationBearing
- bearing_to_vec

3 Core Algorithms

```
def find_blocks()
while current_heading - original_heading < 360 do
    Rotate for one timestep
    update_sensors()
    if object detected then
        check if it's a block or a wall
        if block detected then
            double check in case of error
            if NOT close to (other robot OR home) AND is not within 5cm of another block then
                add block to list
            end
        end
    end
end
if Block list isn't empty then
    Target Block ← closest block
end
```

```
def drive_around_block()
# Set rotation direction towards the centre of the arena by checking the sign of the cross product
if $pos\_vector \times heading\_vector > 0$ then
    rotate $45^o$ CW
else
    rotate $45^o$ CCW
end
Move 30 cm forward

def go(location, range=0.2)
while NOT is_within_range(pos, location, range) do
    rotate_to_location(location)
    if NOT frontClear() then
        drive_around_block()
    else
        go_forward()
        Update timestep
    end
end
```