Activity Bot's Catch Game

Bruno Silva
Github Repository
bruno.silva1@student.um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

ABSTRACT

1 INTRODUCTION

This project is based on 2 robots from *Parallax* the *Activity Bots*, and the goal was to create a recreation of the kids game "The Catch".

There's one robot with a seeker algorithm and another one with a runner algorithm. The seeker uses a pair of ultrasonic sensors to figure which direction an incoming target is coming from. The runner robot is programmed to run around in a pseudo-random way in the playing field in an open loop system.

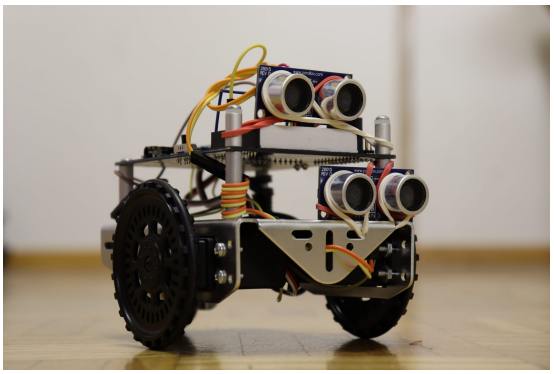

Figure 1: Seeker Robot.

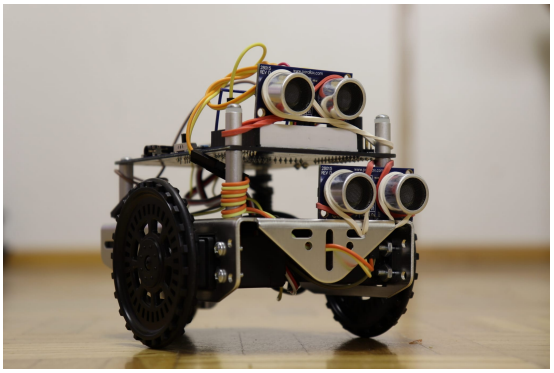

Figure 2: Runner Robot. PLACEHOLDER IMG

2 DEVELOPING ENVIRONMENT

This chapter is meant to explain the working environment of the programming of Parallax's robots and the work flow that was used throughout this project

2.1 SimpleIDE

SimpleIDE is Parallax's official developing platform. As the name suggests, it's a rather simple and feature lacking IDE, meant for absolute beginners. Basic features like autocompletion, keyword highlight, or even error squiggles are not present. This combined with the lack of customization makes a more experienced user rather frustrated when trying to develop on this platform.

Figure 3: SimpleIDE working environment.

2.2 VScode Integration

Fortunately, it is possible, with some workarounds, to migrate the grand majority of the workflow into a more advanced environment like VScode.

(1) PlatformIO
- PlatformIO's IDE is a VScode extension which transforms VScode into a IDE capable of fully programming a wide range of microcontrollers, from code writting to EEPROM flashing. There's a platform there for working the Parallax's microcontrollers (Github). However, upon exhaustive testing this platform seems to be a half finished attempt which doesn't work.

(2) VScode
- One way is to add SimpleIDE's libraries to the default C path for the default C libraries. This approach is better for long term development, but on this project wasn't used.

- Another way is to import the libraries straight into the project's folder, allowing VScode to treat them as user built libraries.
  This was the approach used. To replicate the following folders have to be copied to the project folder:
  - `$ /home/${USER}/Documents/SimpleIDE/Learn`
    `$ /opt/parallax`

  It's important to note that this applies only for Ubuntu based systems and that both these folders should be added to the repository's *.gitignore* as they are quite big and not important to the project itself.

3 SEEKER ROBOT

3.1 Algorithm Explanation

The core of the seeker's algorithm is based on 2 functions:

(1) Assessing targets in range of ultrasonic sensors;
This involves stopping, getting 2 measurements from each sensor spaced by a predefined $\Delta t[s]$, calculating the $\Delta d[cm]$ from the right and left sensor and figure out which direction the target is coming from and outputting accordingly.
(2) Giving moving order to the servos to chase the target. This function purpose is to, given the direction the target crossed from, tune how much the robot should rotate and go forward. This ideally would be dynamically controlled, but since the robot cannot know how fast the target crossed, it has to be manually tuned for a given target velocity.

3.2 Function targetAssessing()

The first thing this function does is assure the robot is still before reading the differential distance measures. The code could account for the velocity of the seeker and correct the measure, but that would be mathematically a lot more challenging.

Then, after reading the distance measurements, the robot could be left with 5 possible case-scenarios given the 2 differential measures :

(1) $\Delta D_L < 0 \wedge \Delta D_R \approx 0$
This would mean the target came from out of range and is started being detected the by left sensor, since the measure distance reduced. The robot should thus turn **RIGHT**;
(2) $\Delta D_L \approx 0 \wedge \Delta D_R < 0$
This would mean the same as case 1, but instead the target is approaching from the right, so the robot should turn **LEFT**;
(3) $\Delta D_L > 0 \wedge \Delta D_R \approx 0$
This would mean the target was already in sight of the left sensor and left suddenly to the left, allowing the sensor to see into the further away background. This would mean the robot should turn **LEFT**;
(4) $\Delta D_L \approx 0 \wedge \Delta D_R > 0$
This would mean the target was already in sight of the right sensor and left suddenly to the right, allowing the sensor to see into the further away background. This would mean the robot should turn **RIGHT**;
(5) $|\Delta D_L| > 0 \wedge |\Delta D_R| > 0$
This would mean that the target is currently in front of the robot and is either getting further away or getting closer. Either way, for the seeker robot the order should be to go **forward**.

In practice, a *threshold* was used instead of 0, to account for sensor measurement error.

3.3 Measuring distances

In order to assure the robot could always read the latest reading from the sensor when needing to evaluate distances, and to take advantage of the parallax's multicore chip, the sensor data is being stored on a global variable that is constantly being updates parallel to the code. The code for acquiring data is running on a separate core.

4 CONCLUSIONS

REFERENCES