

PiCar Mobile Movement Control

Hayden Sierra
Daniel Kelly

Xuan 'Silvia' Zhang, Project Advisor

July 18th, 2018

Table of Contents

Table of Contents.....	1
Abstract.....	2
Introduction.....	2
PiCar Operation Instructions.....	2
Semi-Automated Movement.....	3
Manually Controlled Movement.....	4
Kill Switch.....	5
Data Collection.....	6
Replay Feature.....	6
Known Issues.....	7
Acknowledgements.....	9
References.....	9

Abstract

This report is aimed at introducing new researchers to the findings and progress made on the PiCar Project as of July, 2018 regarding the movement control of the PiCar. This report will cover the currently implemented features of the PiCar, including semi-automated movement, a killswitch, real-time user control using WASD on a keyboard, a replay function, and data collection. Additionally, the last section of the report covers known issues with the PiCar which includes inaccurate servo controls and noisy Inertial Measurement Unit (IMU) data.

Introduction

The PiCar is an easy to build, modifiable, open-source mobile robot platform that is intended to be autonomous. It is controlled with a Raspberry Pi 3 board and an Arduino UNO board, and is equipped with a camera, LiDAR, current sensor, and IMU. Typically, mobile robots like the PiCar suffer from short battery life. If more efficient power orchestration can be achieved in mobile robots, then the battery life of these robots can be extended. The purpose of the PiCar as a research platform is to characterize the power consumption of mobile robots in order to learn how power can be used more efficiently.

PiCar Operation Instructions

Start-Up

1. Upload the Arduino code to the on board Arduino UNO.
2. Make sure everything is hooked up correctly by consulting the schematic.
3. Press button to allow the battery to power the Electronic Speed Controller (ESC).
4. Press the reset button on the Arduino UNO.
5. Immediately switch the ESC on after pressing the reset button on the Arduino UNO.
6. Wait for calibration to be completed (the beeping will stop when calibration is complete).
7. The PiCar is now ready for use. To run a program SSH into the PiCar's Raspberry Pi and run the .py file of choice from the terminal. The .py file must be on the PiCar's Raspberry Pi for this to work.

Shut-Down

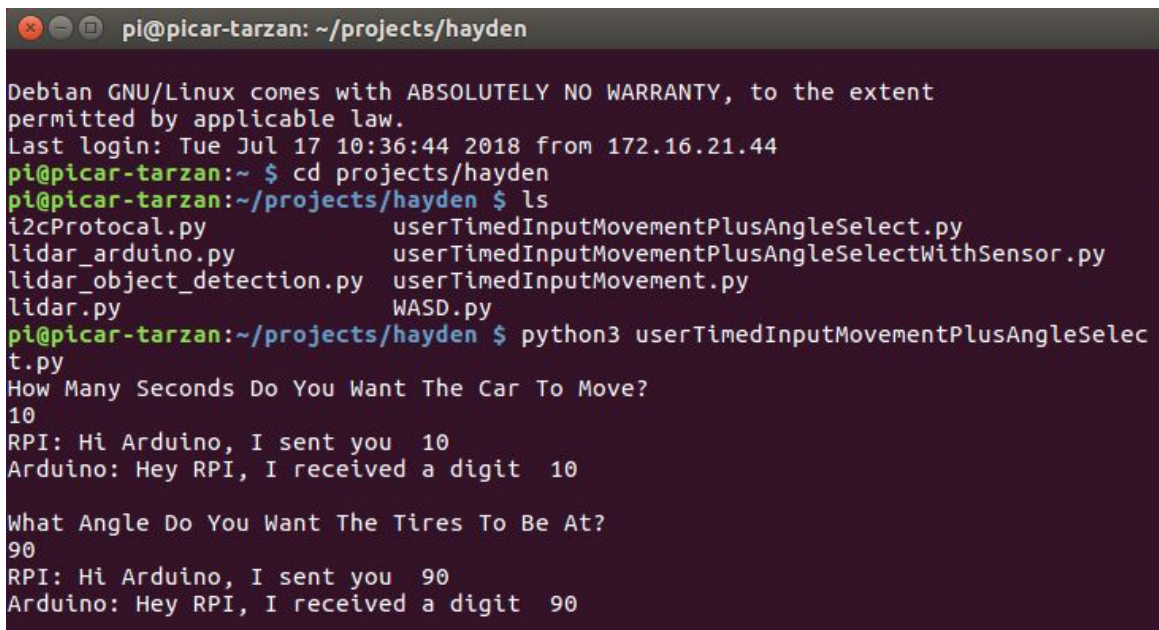
1. Switch the ESC off.
2. Press the button to block power from from the battery.
3. If it's the end of the day, completely unplug the battery from the PiCar.

NEVER UPLOAD ARDUINO CODE WHILE THE ESC IS STILL ON

Note: When uploading code or testing new code, it is generally a good practice to support the car's body in a way that keeps the wheels off the ground.

Semi-Automated Movement

The semi-automated movement of the PiCar involves the user typing in a run time and tire angle, which the PiCar then acts on. The user inputs these values via the command terminal on the PiCar's onboard Raspberry Pi. For example, if the user types in 10 seconds and a tire angle of 68 degrees, the PiCar will go forward for 10 seconds with a tire angle of 68 degrees. A picture of the user interface can be seen in Figure 1.



```
pi@picar-tarzan: ~/projects/hayden
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jul 17 10:36:44 2018 from 172.16.21.44
pi@picar-tarzan:~ $ cd projects/hayden
pi@picar-tarzan:~/projects/hayden $ ls
i2cProtocol.py          userTimedInputMovementPlusAngleSelect.py
lidar_arduino.py        userTimedInputMovementPlusAngleSelectWithSensor.py
lidar_object_detection.py userTimedInputMovement.py
lidar.py                WASD.py
pi@picar-tarzan:~/projects/hayden $ python3 userTimedInputMovementPlusAngleSelec
t.py
How Many Seconds Do You Want The Car To Move?
10
RPI: Hi Arduino, I sent you 10
Arduino: Hey RPI, I received a digit 10

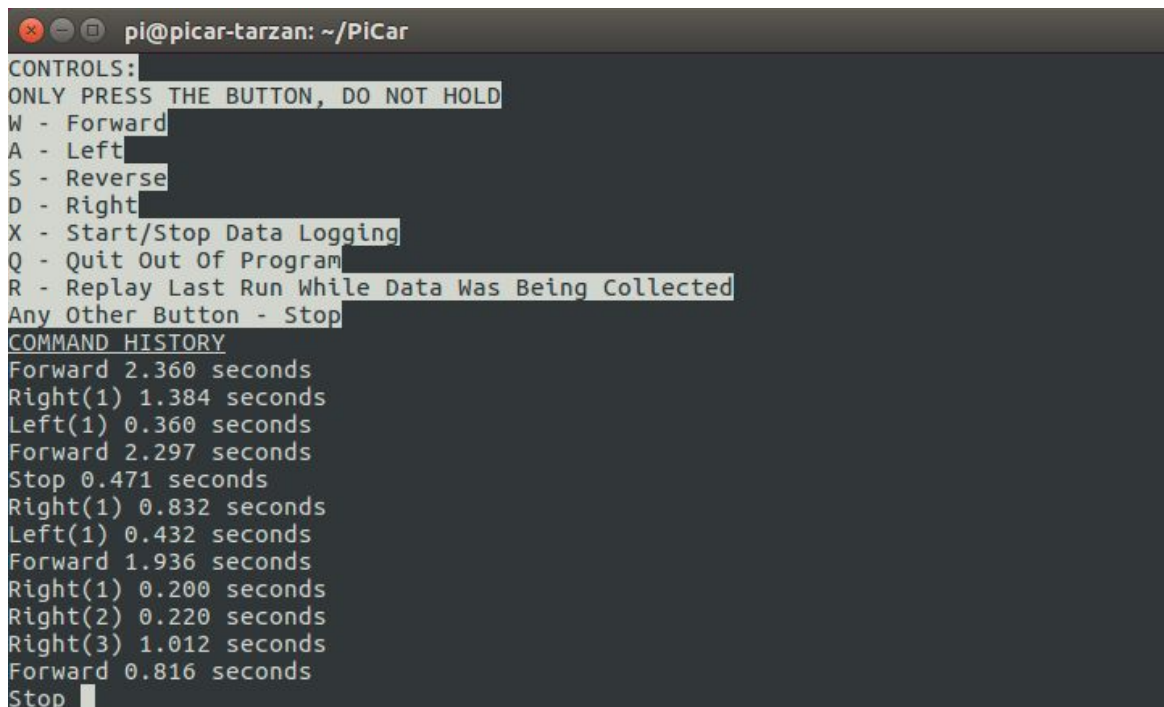
What Angle Do You Want The Tires To Be At?
90
RPI: Hi Arduino, I sent you 90
Arduino: Hey RPI, I received a digit 90
```

Figure 1. The user input control interface for semi-automated movement.

In order for this functionality to be possible on the PiCar, the Arduino code called `semiautomated_movement.ino[1]` must be uploaded onto the Arduino UNO on board the PiCar, and the python code `semiautomated_movement.py[2]` must be executed on the Raspberry Pi on board the PiCar. Both of these code files are available on the PiCar Github. The backbone of the Arduino code for this functionality, as well as any movement Arduino code for the PiCar, is An Zou's `motor_feedback_control.ino[3]` file which is also available on Github. This code makes the PiCar move forward indefinitely at a target RPS. By using this code as a building block, we were able to first develop this semi-automated movement control, but then move on to full real-time manual control using WASD on a keyboard.

Manually Controlled Movement

The PiCar can be manually controlled using the WASD keys on a keyboard. First, the WASD.ino[4] file must be uploaded onto the Arduino. Then using a desktop computer, SSH into the PiCar and run the WASD.py[5] file using the terminal, as seen in Figure 2. Right now the speed is hardcoded in the Arduino script, it is set to target 35 RPS. This can be changed in WASD.ino. Pressing 'w' will drive the car forward. Pressing 'a' will drive the car left, 'd' will drive the car right, and 's' will put the car in reverse. Reverse doesn't work well at all right now. The right and left turns each have three levels. Pressing 'a' once will turn left with the largest turn radius, pressing 'a' a second time will turn left more sharply, and pressing 'a' a third consecutive time will make the car turn sharply left. The same scheme applies for turning the car right with 'd'. To stop the car, simply press any button on the keyboard that isn't 'w', 'a', 's', 'd', 'r', 'q', or 'x'. Pressing 'q' will stop the car and quit the program. Pressing 'x' starts and stops data collection. See the Data Collection section for more details. Pressing 'r' will replay a recorded run, see the Replay Feature section for more details on replaying a recorded run.



```
pi@picar-tarzan: ~/PiCar
CONTROLS:
ONLY PRESS THE BUTTON, DO NOT HOLD
W - Forward
A - Left
S - Reverse
D - Right
X - Start/Stop Data Logging
Q - Quit Out Of Program
R - Replay Last Run While Data Was Being Collected
Any Other Button - Stop
COMMAND HISTORY
Forward 2.360 seconds
Right(1) 1.384 seconds
Left(1) 0.360 seconds
Forward 2.297 seconds
Stop 0.471 seconds
Right(1) 0.832 seconds
Left(1) 0.432 seconds
Forward 1.936 seconds
Right(1) 0.200 seconds
Right(2) 0.220 seconds
Right(3) 1.012 seconds
Forward 0.816 seconds
Stop
```

Figure 2. The user input control interface for real-time manually controlled movement.

This code works by continuously expecting keyboard input. This is achieved with a loop. Once input is received, if/else statements determine what happens next. If a keystroke for movement is entered ('w', 'a', 's', 'd', or any other key that doesn't have an explicit function), a number that is uniquely associated with that keystroke is passed from the Pi to the Arduino using

I2C. In the Arduino, there is a switch statement in the loop() method. The last value passed to the Arduino from the Pi is continuously fed into the switch statement. For example, if 1 is passed to the Arduino, until a new value is passed the Arduino code will continuously loop through the 'case 1' code, which translates to the car continually driving forward until a new input is entered. If a keystroke that isn't associated with movement is entered, like 'x', the car will continue behaving as it was before. If a key that doesn't explicitly have a function is entered, the default case in the switch statement will run, which stops the car.

A quick note about steering the car: the servo motor that controls steering does not respond well to small changes, and it does not respond well in the middle ranges. As a result, when trying turn from the far left (a servo position of 120) to the center (a servo position of roughly 90), the current procedure is to turn from 120 to 75 to 90. If turning to center from left with the wheels on the ground, 75 gets the wheel straight ahead. Going to 90 from 75 is mostly about bookkeeping, the wheels hardly move. For turning the wheels to center from right, writing the servo position to 105 gets the wheel straight. These will likely change with the weight of the car. Currently in addition to the batter there is a power bank that is used to power the Pi and Arduino. If future designs power everything from one power supply and the weight of the car is reduced, these values for getting to center may have to be recalibrated.

Kill Switch

As a safeguard against unpredictable behavior of the car (for example, if it accelerates to its maximum speed and can no longer be controlled), a kill switch feature was implemented to prevent crashes. The problems often stem from the ESC. If the Arduino crashes and ceases providing instructions to the ESC, the default seems to be to drive the motor at full throttle. The kill switch is a physical switch that shuts off power to the ESC, which in turn shuts off power to the motor. **To open the switch and stop the car, simply turn on the remote controller pictured in Figure 3. To use the car again, the remote controller must be turned off, and the Arduino must be reset.** When the remote controller is turned on, it sends a signal to the receiver, which is connected to the Arduino. When the receiver gets a signal from the controller, an interrupt in the Arduino script is triggered. This interrupt writes pin 7 on the Arduino from HIGH to LOW. This pin is connected to the relay seen in Figure 4, such that when the pin output is LOW, the relay is opened and the ESC receives no power. Note that the signal from the Arduino to the relay only opens and closes the switch that connects the battery to the ESC. The ESC is powered by the battery. When the relay switch is closed, the red light is on and the green light is off, as in Figure 4. When the switch opens, the green light turns on too.



Figure 3. The remote controller that operates the kill switch.

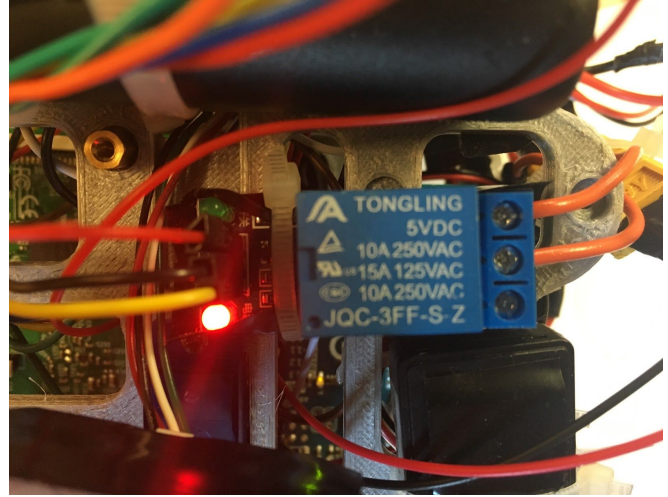


Figure 4. The relay that switches off power to the ESC.

Data Collection

The data collection functionality of the PiCar works by utilizing the data collection code written by Jerry Kong[6]. An explanation of how the code works in detail can be found in the Software section of the readthedocs[7]. The way it has been implemented with the mobile movement control is by pressing ‘x’ on the keyboard when the user wants to collect data from the IMU, camera, LiDAR, and current sensor. The user can then press ‘x’ again to end the data collection process. The data collection code can be edited to make it so the data is automatically stored on the PiCar itself, or it requests a data transmission protocol to send the data to another computer wirelessly.

Replay Feature

The replay function is built in to the WASD.py file. It replays a previously recorded run. To record a run, from the WASD.py terminal press ‘x’. The ‘x’ key records data using the sensors, but it also records the keystrokes entered and the interval between keystrokes. After a run has been recorded, pressing the ‘r’ key will input the same keystrokes from the last run with the same intervals between those keystrokes. Only one run at a time can be recorded for replaying with the current implementation. The current implementation works fairly accurately with short, simple runs, but with longer and more complex runs the replay function becomes very inaccurate. This is a result of the current implementation of the replay function being completely open loop. Small errors compound over time and there is no feedback to check this error. A version of this feature that incorporates feedback from the IMU is in the early stages of

development right now, but it is at this time uncertain how the IMU data will be used. It can be very noisy.

Known Issues

One of the most persistent problems with the PiCar is that the servo that controls the tire angles has inaccurate fine adjustment, meaning that if you go from 60 degrees to 65 degrees nothing will change. This especially becomes a problem when setting the car to go straight when it was going left. For some reason the servo keeps the car going left even when you set the servo to 95 degrees which would normally make the car go straight when coming from any other direction. After much testing, it was concluded that this problem is most likely a hardware restriction that will just have to be worked around. The solution in place for now is to make the servo go from the left to the right and then straight in order to simulate it going from right to straight. This makes the car go straight when coming from the left as it should be.

The other main problem is the noisy IMU data. The IMU that is installed is not state of the art, thus it is not the most accurate IMU ever. The inaccuracies are made even worse when coupled with the fact that the car is constantly vibrating when moving. Figures 5, 6, and 7 showcase the noise of the accelerometer, gyroscope, and orientation readings.

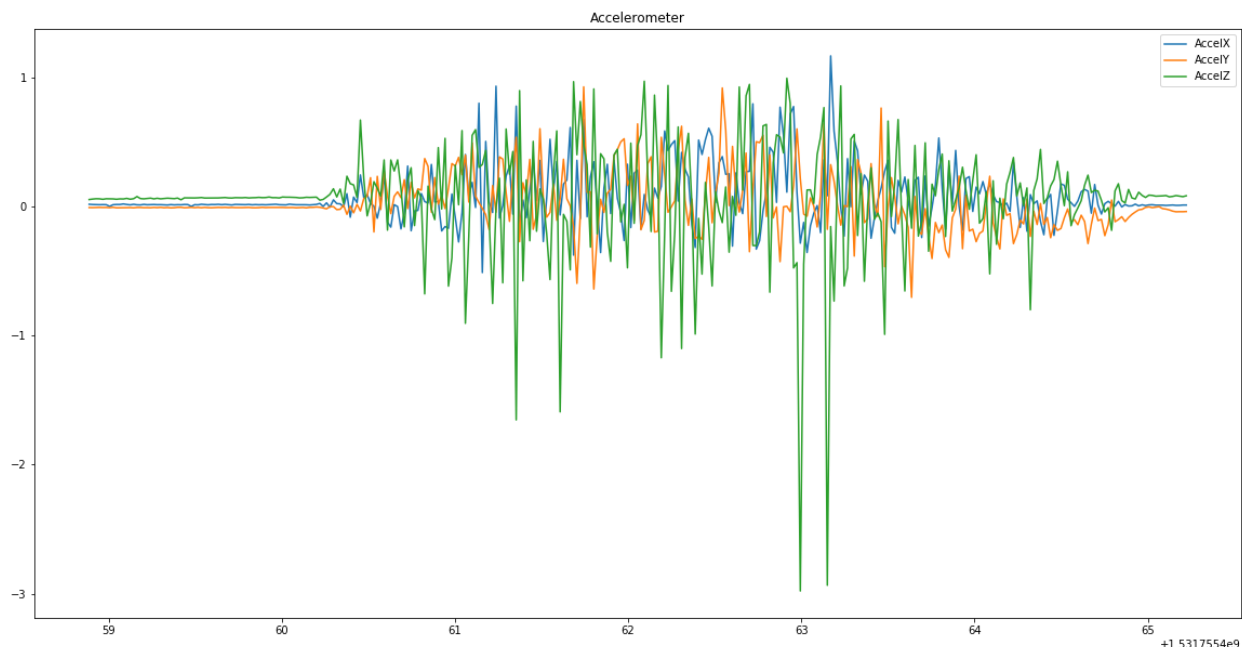


Figure 5. Accelerometer data collected by the IMU.

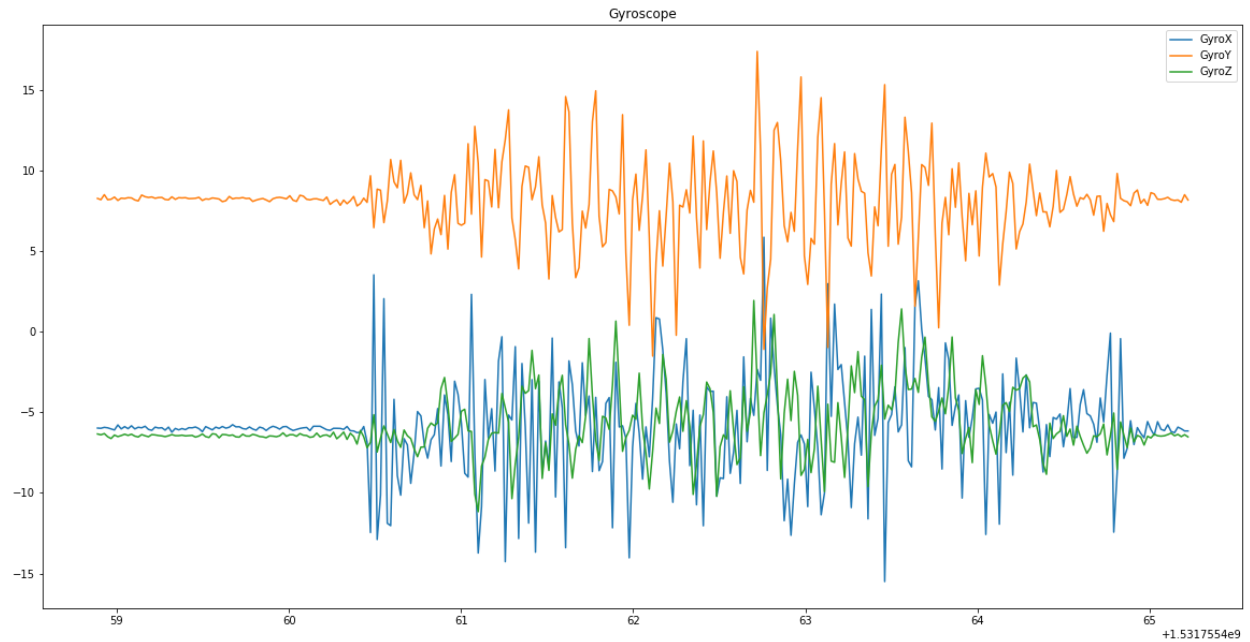


Figure 6. Gyroscope data collected by the IMU.

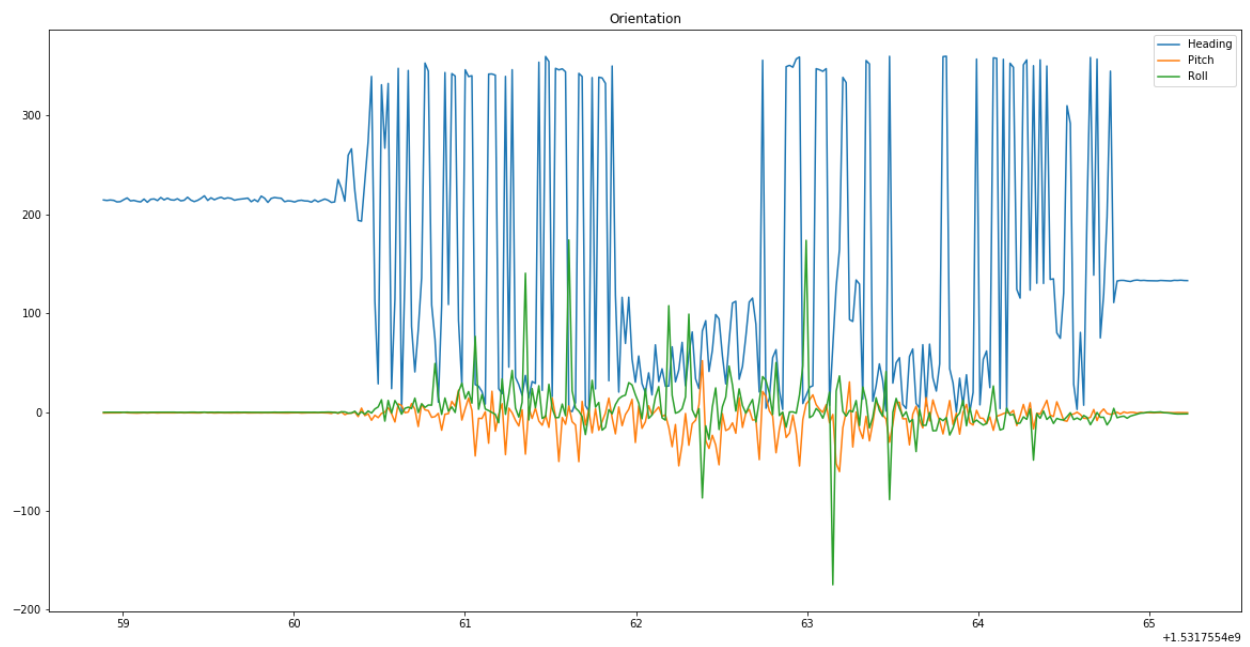


Figure 7. Orientation data collected by the IMU.

Attempts are being made to smooth out the noise by passing the data through a filter, but so far there have been no accurate results. Figuring out how to get reliable data from the IMU will be a big breakthrough when it comes to using the data as feedback for true autonomous navigation.

Acknowledgements

We would like to thank Professors Xuan ‘Silvia’ Zhang and Christopher Gill for mentoring us through this project and keeping us on track. Also we’d like to thank Adith Bloor and An Zou for always being there to help us in the lab, and always teaching us something new. Lastly we’d like to thank our other team members Karina N. Martinez-Reyes, Shadi Devari, Feiyang Jin, and Jerry Kong, whose work and help we’d be nowhere without.

References

- [1] semiautomated_movement.ino
https://github.com/xz-group/PiCar/blob/master/src/mobile_movement_control/semiautomated_movement.ino
- [2] semiautomated_movement.py
https://github.com/xz-group/PiCar/blob/master/src/mobile_movement_control/semiautomated_movement.py
- [3] motor_feedback_control.ino
https://github.com/xz-group/PiCar/blob/master/src/arduino/motor_feedback_control/motor_feedback_control.ino
- [4] WASD.ino
https://github.com/xz-group/PiCar/blob/master/src/mobile_movement_control/WASD.ino
- [5] WASD.py
https://github.com/xz-group/PiCar/blob/master/src/mobile_movement_control/WASD.py
- [6] Jerry Kong’s Code https://github.com/xz-group/PiCar/tree/master/src/pi/IMU_Lidar
<https://github.com/xz-group/PiCar/tree/master/src/logging>
- [7] readthedocs <http://picar.readthedocs.io/en/latest/>