That's so Fetch

Luca Amblard, Dan Barychev, Hana Frluckaj
Electrical and Computer Engineering,
Carnegie Mellon University
{lamblard, dbaryche, hfrlucka}@andrew.cmu.edu

Abstract

Our project is to develop a motorized device that plays Fetch. After an initial calibration with the user, the device will be able to move away, anticipate the trajectory of the object thrown using a pre-throw, catch the object thrown, and move back to the user to return the object. Calibration will be facilitated through the use of IMUs, which are also instrumental in predicting object trajectory from the user's hand. Communication from user to device is made through wireless connection between a Particle Photon and a NVIDIA Jetson Nano.

Index Terms

Inertial Measurement Unit (IMU), Particle Photon, I2C, Wireless Communication, NVIDIA Jetson Nano, Kalman Filter, Dead Reckoning, Target Estimation, PID

Introduction

Our project is aimed towards children who would like to play fetch with a real dog but are unable to because of allergies, housing constraints, and other external factors. While the only current function of our device is the ability to play fetch (i.e. anticipate throw, catch, and return), most commercial robot dog products do not have this option. As far as other motorized devices capable of catching items, there have been similar projects in the past, e.g. Smart Trashbox from Minoru Kurata and Smart Trash Can from F19 Team B4. Kurata's device had a success rate of about 20% but was highly adaptable and motorized, while Team B4's device had a success rate of about 50% with small movements and low motorization (i.e. small range of ~30 cm). Our goal is to create a highly mobile device (i.e. range of 1m) with a success rate over 50%. While previous projects had different approaches, to our knowledge, this is the only project to use inertial measurement units to predict the trajectory of the object thrown. Our method involves an initial calibration between user and device, data sent over wifi to the motorized device, a control system that coordinates the omnidirectional wheels to anticipate the landing, and a successful return of the caught object to the user.

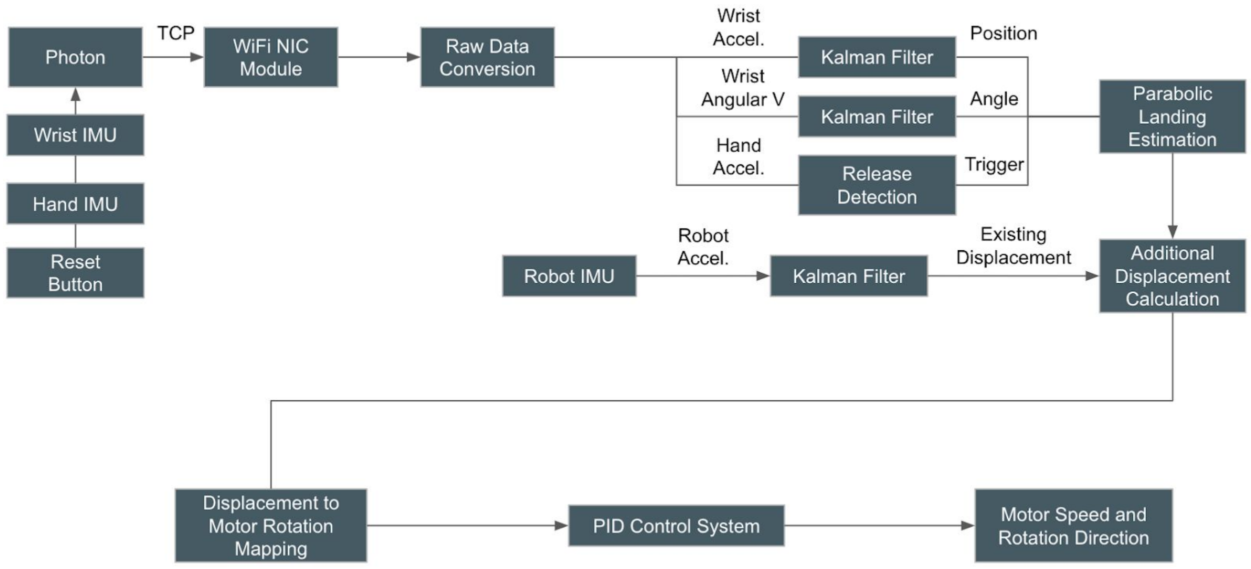
Design Requirements

The design requirements for our catch-and-retrieve system center around the need for an intuitive dog simulator that can catch at a >50% rate and effectively bring the ball back to the user for consecutive throws to different locations. In order to achieve a >50% success rate and properly mimic the experience of playing catch, we are enforcing a starting distance of 2m away from the user and limiting the catch radius to 1m around this starting position. Thus, if the user decides to throw outside of this range, the simulator will refuse to catch the ball and a manual reset will be necessary.

In order to create a viable catch scenario, we are also enforcing the use of an action we term the "prethrow". This forces the user to wind up into their next throw and the arc of this prethrow thus indicates the approximate landing location and angle. Once the simulator is facing the user, the user can release the ball and the robot's adjustments towards the actual landing location can take place. These adjustments are thus likely to be minimal and realizable within the time it takes for the ball to be caught. We also ask that the user throw the ball in the general direction of the robot once the robot is facing them. This minimizes the need for lateral movement which is generally slower than vertical movement. To make sure that the angle of the prethrow and actual throw don't significantly differ, we are setting a requirement for a <5% difference between the two that we will test by subtracting them as soon as the actual throw occurs.

Another critical aspect to our design is the maintaining of correct relative distance between the simulator and the user. We will begin the throw experience by the user picking the ball out of the catch receptacle to calibrate position between their tracking IMU and the robot's tracking IMU. Then, after the simulator drives out 2m, catches the ball, and returns to the user, the user's retrieval of the ball should recalibrate the position system. However, it is more than likely that a positional offset can occur during this time. Thus, we are enforcing a requirement of less than 12.5cm offset per each catch since 12.5cm will be the radius of the catch receptacle. We will calculate this offset by measuring the distance away from the proper 2m drive-out point at the start of the next catch. If this requirement is not met, the user will have to place the robot in front of them and initiate a physical reset with the IMUs in extremely close proximity.

Architecture and/or Principle of Operation



Our design consists of two main parts: the IMU data collection and transmission from the user's arm and the robot which is responsible for all the computation and reaching the target location on time to catch the ball thrown by the user.

We will be using the MPU 9250 (IMU) for motion sensing. It is small, light and contains an accelerometer as well as a gyroscope which we will both need to estimate the location where the ball will land. The raw data of the devices is read by the Particle Photon which communicates with the IMUs through the I2C serial communication protocol.

The Particle Photon, which is an IoT device, transmits the IMU raw data to the Jetson Nano which is located on the robot car chassis. All the computation is done on the Jetson Nano since it has more processing power than the Photon. The Particle Photon and the Jetson Nano communicate via WiFi. When the whole system is turned on, a socket connection is set up between the two devices and the Particle then sends data streams of IMU data to the

Jetson Nano via the Transmission Control Protocol (TCP).

Once the accelerometer and gyroscope data have been received on the Jetson Nano they are converted to $m\ s^{-2}$ and $deg\ s^{-1}$ respectively. The data is then passed through a Kalman filter which is used to obtain 3D points in space. The angle of projection and speed of the ball when it is released from the hand are applied to suvat equations to predict the landing location of the ball. Details about how the ball release from the hand is detected will be covered in the System Description section. We'll have to use the heaviest possible ball that will not break the robot in order to minimize air resistance. The exact size and weight will be determined when we receive the robot cart chassis.

As the robot moves the distance left to travel and the direction to the target are updated using the previous position and the distance and direction travelled from that position. The data of the displacement from the previous position is obtained by passing the data of the IMU on the board through Kalman filters. The Jetson Nano communicates with the IMU on the robot using I2C as well.

The robot moves by controlling four motors on an omni-directional robot car chassis. To move at a given angle the robot car will have to move each of the four motors at certain speeds and angles. We will tune the parameters for mapping angle to motor target speeds and direction when we receive the robot car chassis. The direction and speed of each motor are fed into a PID control system that determines the Pulse Width Modulation (PWM) duty cycle. This duty cycle is passed into the PWM driver and the direction of rotation is passed into the motor driver. The speed of the given motor is determined by the encoder driver. The error, which is the absolute value of the difference between the target speed and actual speed are passed back into the controller of the system.

We decided to use C++ for the following three reasons: C++ is compatible with the Jetson Nano, C++ is the language used to program the Particle Photon so this would make the networking between the devices easier and C++ is one of the fastest programming languages, which is important for the project because the robot needs to react very quickly.

Design Trade Studies

Tradeoffs over IMU placement (inside ball v. on wrist)

In discussing possible IMU placement configurations, we considered the possibility of placing an IMU inside the ball. The benefit to doing this would be the ability to track the trajectory of the ball as it moves through the air, thus mimicking the functionality of a CV setup. The issue with this setup, however, is that we would not know the ball's release point from an internal IMU alone. In order to avoid beginning landing prediction once the ball fell from the air, we would still need to have an IMU on the user's hand to determine release. We decided that having IMUs in three different locations (hand, ball, and robot) would be too excessive and would take away from the uniqueness of the technical challenge. In addition, we would like for our system to work with multiple types of balls. We thus decided on having a knuckle IMU for release and a wrist IMU for trajectory upon release.

Jetson Nano v. Raspberry Pi

We are using a Jetson Nano in our project as opposed to a Raspberry Pi because it has more processing power so it would be able to do computation faster. Cost is not a factor because Luca already owns a Jetson Nano. There are no device compatibility issues that arise in our project from the use of the Jetson Nano instead of the Raspberry Pi.

Photon v. Arduino & Transmitter

We needed a WiFi transmitter in order to send data from the IMUs to the Jetson, and the two best options were just using a Photon, or using an Arduino in conjunction with a transmitter. We decided to use the Photon for a number of reasons. Hana was already familiar with the device since they had worked with it on a previous project. There is an active community surrounding the device with detailed documentation. The device also has a web IDE and intuitive user interface. In the event that we have irreconcilable issues with the Photon, we may switch to an Arduino with a transmitter (and possibly a receiver), although that may lead to some latency issues.

I2C v. SPI

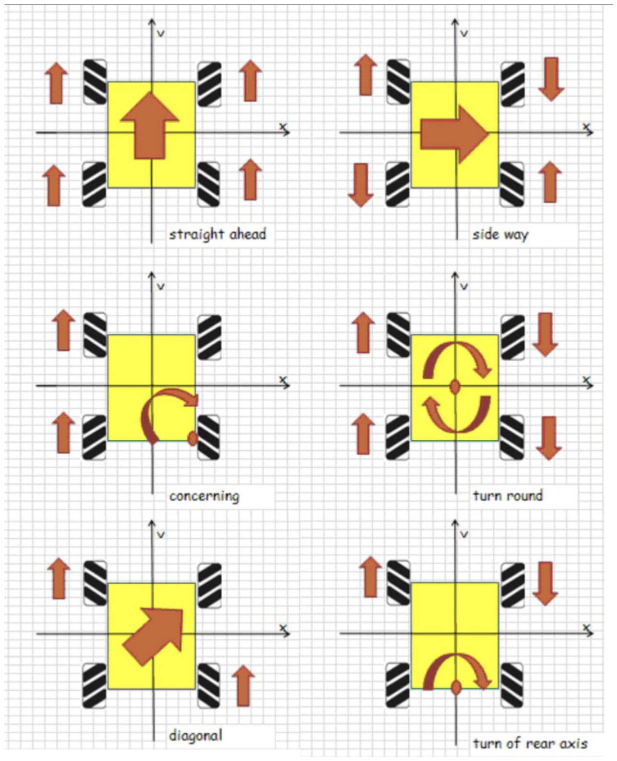
The the main advantages of SPI over I2C are the following: SPI is faster and consumes less power than I2C. However, we have decided to use I2C for the serial communication between the arm IMUs and the Particle Photon and the IMU on the robot and the Jetson Nano. I2C is less affected by noise than SPI and makes sure that data sent is received by the target slave device. Receiving accurate data from the IMUs reliably is important to accurately predict the ball's landing location coordinates. Additionally, I2C is better suited to send data through long wires. The wires would be along the user's arm in our case.

TCP v. UDP

One thing when planning communication over a browser socket between the Photon and Jetson is that there are two options: UDP and TCP. Both are protocols used for sending packets over the Internet and both are built on top of the IP protocol. TCP is much more commonly used, and UDP is mostly the same except that it doesn't do error correction, resulting in faster speeds. If we were worried about speed, we might consider UDP. However, given the extra time from the pre throw and the Jetson's overkill processing power, TCP is the better choice. There are more resources online for sending data through TCP and error correction can benefit us in debugging and other issues down the road. [8]

Omni-Directional Wheels

We are using omni-directional wheels for the robot so it can directly move to the target location in a straight line. We think that this method would be faster than rotating the robot and then moving in the direction of the target location. It is crucial for the robot to reach its destination as fast as possible in order to catch the ball. We have not received the Mobius robot car chassis yet but we will proceed to testing both approaches to confirm this choice as soon as we receive it. The diagram below indicates how omnidirectional wheels can seamlessly change direction.

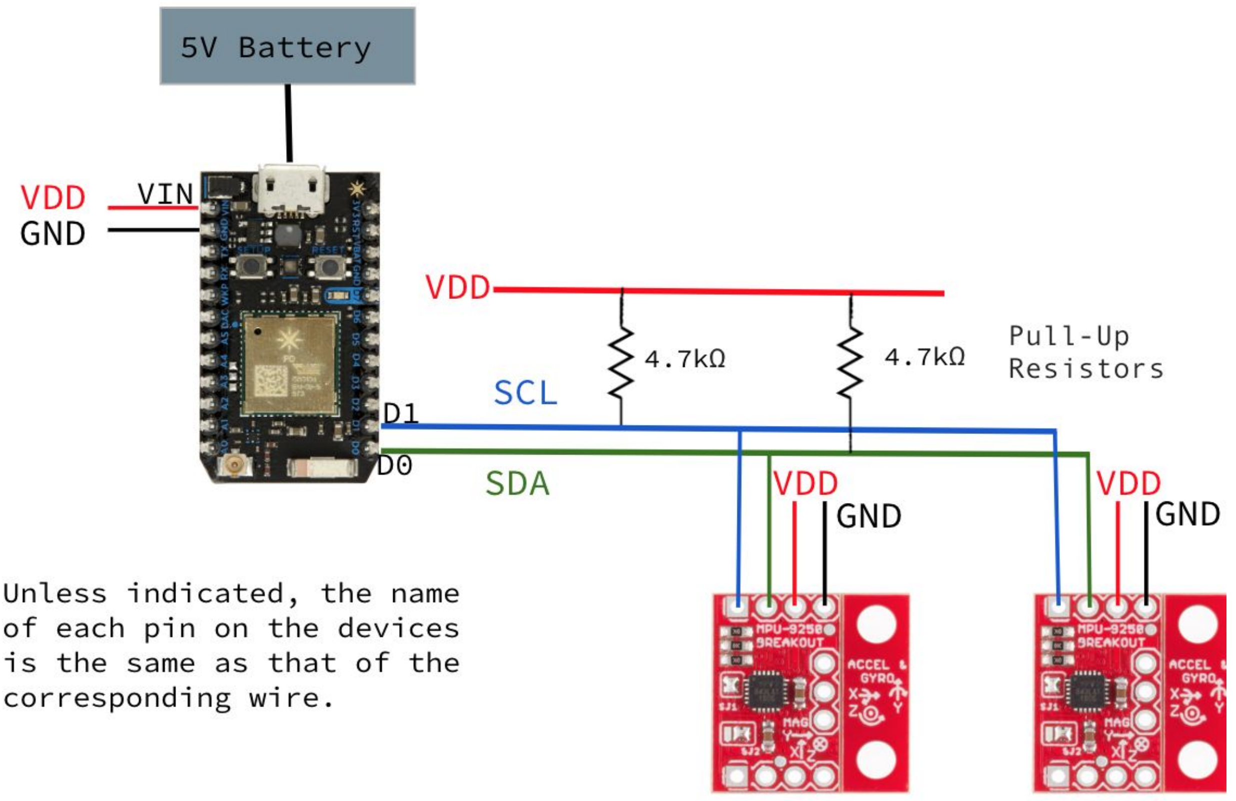


WiFi v. Bluetooth

In our choice for a WiFi module, we discussed the differences between the Particle Photon, a WiFi IoT device, and the Particle Argon, which is also Bluetooth capable. In the end, we decided that WiFi best fits our constraints. We're planning on having the device capable of catching over a relatively large distance compared to other projects, 2 meters. WiFi is much better at covering this distance at a much faster speed. A drawback of WiFi is latency and timing issues; however, this issue is eliminated by the fact that our method involves the pre throw. We timed the transmission of data over WiFi, which turned out to be 0.1 seconds. This is well within the time granted to us from the end of the pre throw's trajectory at the cusp of release to the time for the user to bring their arm back for the actual swing, and then for the user to actually swing and throw the object, about 2 to 3 seconds. Therefore, we're confident WiFi is the best choice for our project.

System Description

IMUs to Photon


5V Battery
VDD
VIN
GND
VDD
4.7kΩ
4.7kΩ
Pull-Up Resistors
SCL
D1
D0
SDA
VDD
GND
VDD
GND
Unless indicated, the name of each pin on the devices is the same as that of the corresponding wire.

The gyroscope data and accelerometer data are read from the two MPU 9250 devices (IMUs) by the Particle Photon using I2C. The I2C clock frequency for the IMUs is 400kHz. The data is then transmitted from the Particle Photon to the Jetson Nano over WiFi. The Photon and the two IMUs are connected as shown in the diagram above. We found a MPU9250PhotonLibrary [1] which contains code to initialize I2C as well as code for the Photon to read from and write to different IMU registers.

The IMUs contain a register called WHOAMI which contains the device address of the IMU. One IMU will have WHOAMI register value of 0x71 which is the default value and another IMU will have its WHOAMI register value changed to another value so the Photon can address each individual IMU.

We will first try 4.7kΩ pull-up resistors as these are typical pull-up resistor values to start with. We will carry out some tests to determine the highest possible pull-up resistor values for which the I2C serial communication will function properly in order to minimize current flow through them and power consumption. The Photon has an internal pull-up resistor but the IMUs don't so we will just use these pull-up resistors shown in the diagram for both the Photon and the IMUs.

The gyroscope configuration and accelerometer configuration registers will be used to select the full-scale range for each device. The gyroscope can be set to have a full-scale range of either ± 250dps (degrees per second), ± 500dps, ± 1000dps and ± 2000dps. The accelerometer can be set to have a full-scale range of either ± 2g (g = $9.81ms^{-2}$), ± 4g, ± 8g or ± 16g. We will confirm the scale to use for each device by spending time moving the IMUs in different ways while data is being collected from them. It is important to use the smallest scale possible for each device because smaller scale range leads to greater sensitivity.

The IMUs will be polled for data at a frequency of 100Hz. The reason for this is the following: the polling frequency needs to be high enough to detect the user's hand opening, which is when the ball leaves the hand. Five IMU readings are required during the time that the hand is opened and our estimate is that it will not take less than 0.05s for a user to open their hand. To poll the IMUs five times in 0.05s, we need a polling frequency of 100Hz.

Particle Photon to NVIDIA Jetson Nano

The Particle Photon is not only capable of receiving data over WiFi, but it is also a WiFi transmitter. We will be using this to send data to the NVIDIA Jetson Nano, which is capable of receiving such data. The Photon has a single antenna port and with a frequency band of 2.412 to 2.462 GHz. When the Photon is powered via the USB port, VIN will output a voltage of approximately 4.8VDC. When used as an output, the max load on VIN is 1A. The typical average current consumption for it is 80mA with 5V at VIN with Wi-Fi on. [10]

The NVIDIA Jetson Nano has serial peripheral interface controllers that operate up to 65Mbps in master mode and 45Mbps in slave mode. It allows a duplex, synchronous, serial communication between the controller and external peripheral devices. It consists of four signals, SS_N (Chip select), SCK (clock), MOSI (Master data out and Slave data in) and MISO (Slave data out and master data in). The data will be transferred through MISO on every SCK edge. The receiver always receives the data on the other edge of SCK.

The goal is to create a serial connection between the Photon and the Jetson, hopefully achieved by sending a message virtual serial port which is then redirected to a portal using the TCP protocol. The Photon will act as the TCP Server and the Jetson as the TCP Client, with the Jetson needing a Nodejs server that can send "D7-ON" and "D7-OFF" and the Photon needs a means to communicate with it, which we are planning on using a modern browser web socket for. [11]

IMU Raw Data Conversion

Here are the formulas to convert gyroscope data and accelerometer data to angular velocity(in $deg\ s^{-1}$) and acceleration (in g) values respectively:

For the gyroscope:

angular velocity (*dps*) =
raw data / sensitivity scale factor

For the accelerometer:

acceleration (g) =
raw data / sensitivity scale factor
g: gravitational field strength in $m\ s^{-2}$

The raw data can take values between 0 and 32768. Sensitivity scale factor (S.F.) is determined by the full-scale range used by the corresponding device. The details about full-scale range are located in the IMUs to Photon section. The tables below show the sensitivity scale factor that corresponds to each full-scale range for each device (accelerometer and gyroscope).

Gyroscope :

Full Scale Range ($deg\ s^{-1}$)	Sensitivity S.F. ($deg^{-1}\ s$)
± 250	131
± 500	65.5
± 1000	32.8
± 2000	16.4

Accelerometer:

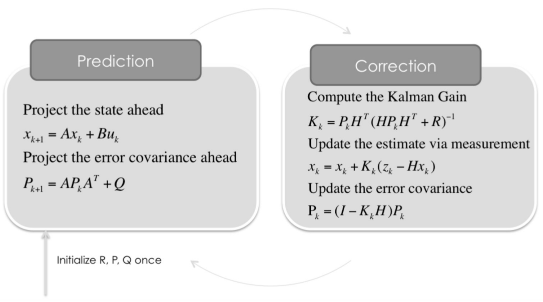
Full Scale Range (g)	Sensitivity S.F. (g^{-1})
± 2	16384
± 4	8192
± 6	4096
± 8	2048

Kalman Filters

The key to deciphering the position across our catch system will be the Kalman filter. This mathematical concept allows us to determine approximate position, velocity, and angle by using successive measurements of acceleration and angular velocity from our IMUs. The key to the Kalman filter's functionality is a continuous loop between predicting the upcoming state and updating the error covariance calculations we use to create that prediction. As a result, we receive a relatively accurate representation of our position by taking into account the error that occurs due to the IMUs constant motion.

We will use a Kalman filter to determine the 3D position, velocity, and angle of the ball upon its release point. The wrist IMU's acceleration readings will be used to predict both velocity and position. The IMU's angular velocity readings, on the other hand, will be used to determine the release angle.

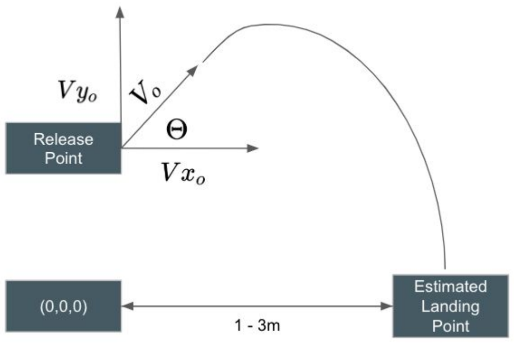
The Kalman filter will also be used to track the positional movement of the simulator from its IMU. These readings will be essential to maintaining the correct relative position between the user and simulator.


Prediction
Project the state ahead
$x_{k+1} = Ax_k + Bu_k$
Project the error covariance ahead
$P_{k+1} = AP_kA^T + Q$
Correction
Compute the Kalman Gain
$K_k = P_kH^T(HP_kH^T + R)^{-1}$
Update the estimate via measurement
$x_k = x_k + K_k(z_k - Hx_k)$
Update the error covariance
$P_k = (I - K_kH)P_k$
Initialize R, P, Q once

[12]

Parabolic Landing Estimation

Once we receive the 3D coordinates, speed, and angle from the Kalman filters, we must perform landing estimation using common parabolic equations. The necessary definitions are shown in the diagram below



$$Horizontal\ distance,\ x = V_x t$$

$$Horizontal\ Velocity,\ V_x = V_{x_o}$$

$$Vertical\ distance,\ y = V_{yo}t - \frac{1}{2}gt^2$$

$$Vertical\ Velocity,\ V_y = V_{yo} - gt$$

We will derive t from the equation for vertical distance and then apply it to the equation for horizontal distance. Since we are defining the robot's movement in terms of vertical distance and angle (similar to polar coordinates), we don't need to project the horizontal distance equation onto another plane.

Displacement Update and Motor Rotation Mapping

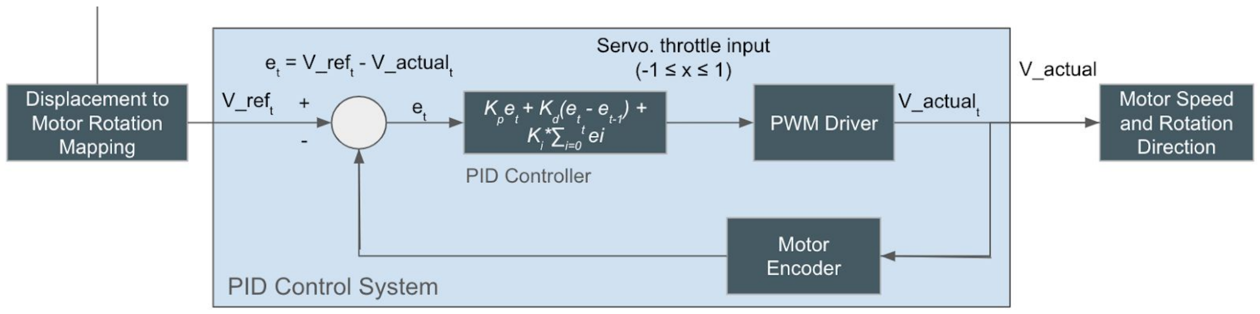
The robot's projected travel distance and direction are updated at a regular frequency, which we will determine, using data read from the IMU on the robot. Once the final landing place is anticipated, the robot now has a fixed direction to move in. This direction determines in which manner the omnidirectional wheels will spin, albeit all in a uniform manner. The mapping from direction and distance to travel to motor rotation will be determined once we receive the robot car chassis. From the IMU data received at a regular interval, we may obtain how close the robot is to the fixed landing location, and as a result, control acceleration/ deceleration.

Motors

The NVIDIA Jetson Nano will also be used to control the motors on the Moebius omnidirectional wheel base. The motors within the Moebius base are servo motors. Our goal is to control the servo motors over an I2C-controlled PWM/servo driver. PWM is a common method used for controlling motors because of fast digital pulses instead of a continuous analog signal. The Jetson has the capability of generating two PWM pulses on the J41 Header. However, we need to reconfigure the device tree to make these signals available on Pins 32 and 33. We will be using the PCA9685 breakout board, which uses 3.3V supply from J21 Pin 1, and GND from J21 Pin 6.

We will be using the Adafruit ServoKit Library because it is a CircuitPython helper library for the PWM built on top of the Jetson Library, which also has a C++ version. We're going to be using the version that supports C++ as our other devices, the Photon and Jetson, also use C++. A 5V and 4A power supply is connected to the breakout board. From the library, we will be using the '*throttle*' command function, which is useful for controlling continuous servos and the similar electronic speed controllers for motors.

The rotation of the servos will be controlled using a PID control system. The procedure for each servo is as follows. The target speed is input into the control system as shown in the diagram. The error is calculated by subtracting the target speed from the actual measured speed and it is input into the PID controller which gives a value between -1 and 1. This value is input into the servo.throttle function mentioned above and this causes the servo to rotate. The actual speed of rotation of the servo is measured by the servo encoder and is fed back to calculate the new error. The proportional, derivative and integral parameters will be tuned to minimize steady-state error, rise time and settling time. [13]


e_t = V_ref$_t$ - V_actual$_t$
Servo. throttle input
(-1 ≤ x ≤ 1)
V_actual
Displacement to Motor Rotation Mapping
V_ref$_t$
+
-
e_t
$K_p e_t + K_d(e_t - e_{t-1}) + K_i * \sum_{i=0}^{t} ei$
PWM Driver
V_actual$_t$
Motor Speed and Rotation Direction
PID Controller
Motor Encoder
PID Control System

Project Management

Schedule *(see last page)*

Team Member Responsibilities

Currently, Dan is detailing the Kalman filter design to map raw IMU data to coordinates in 3D space, Luca is focused on the intricacies of the I2C communication between the Particle Photon and several IMUs, and Hana is dealing with the networking between the Particle Photon and the Jetson Nano and also the motor drivers on the Jetson Nano. In future steps, Dan will be largely responsible for projectile prediction, Dan and Luca will be responsible for the controls aspect of the base, Luca will be primarily responsible for IMU data analysis and system architecture, Hana and Luca will work together on interfacing the Photon with IMU data and sending that to the Jetson, and Hana will be responsible for motor control and base construction. All team members will convene towards the end of the project for ensuring proper retrieval and returning of the device.

Budget

Date	Item	Cost	Total Left
1 Feb	NA		$600
8 Feb	6 IMUs and shipping (SparkFun)	$89.30	$510.70
10 Feb	1 Photon and shipping	$27.81	$482.89
17 Feb	Moebius Chassis	$63.99	$418.90

Risk Management

Currently, we are still waiting on the arrival of the Moebius omnidirectional wheel base. It was estimated to arrive this past week, but recent issues indicate that our package is lost. We are not sure when it will arrive, so we are adapting to this by working on all other aspects.

There are several aspects of our project which are dependent on communication between different modules (i.e. IMUs to Photon, Photon to Jetson, Jetson to servo motors). We will do preliminary tests to just send a few bits of data from each, get acknowledgements, and send some data back. This will ensure that communication is possible and does not hinder further work that depends on communication as a foundation.

Related Work

There are two existing similar projects: Smart Trashbox from Minoru Kurata and Smart Trash Can from F19 Team B4. The success rate for both projects were relatively low, with Kurata's device at about 20% and Team B4's at about 50%. Clearly, we hope to accomplish a success rate over 50%.

Kurata's device is a much faster and mobilized version than Team B4's, and so it could cover greater distances. This probably contributed to the low success rate there was a lot of distance to cover. From Team B4's project demo, it appears that their trash can could not move very far (~30cm), and so the team had a high success rate from aiming the object thrown very close to the device's rim. We hope to make our device much more receptive.

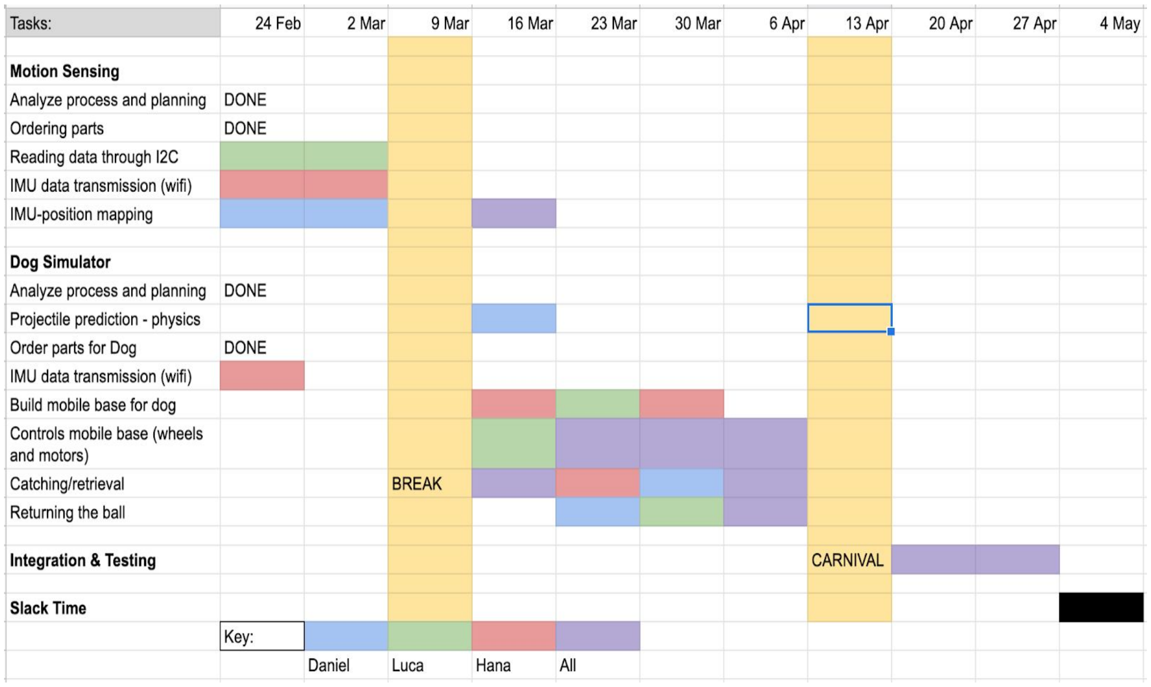
Summary

Our goal is to create a motorized device that is able to play the game of Fetch, hence the infamous title of 'That's so Fetch.' Our project starts with an initial calibration with the user where the IMU on the user's hand is close to the the device's IMUs for a starting point. Then, the device will be able to move away at a distance about 2m from the user, anticipate the trajectory of the object thrown using data from the user's pre-throw, catch the object thrown, and move back to the user to return the object. Moving back to the user is possible by reversing the difference from the initial calibration. Communication from user to device is made through wireless connection between a Particle Photon and a NVIDIA Jetson Nano. The Jetson will then control the servo motors, which guide the omnidirectional wheels, and move them in a way where our 3D printed net atop the base will catch the object.

References

[1] Velasquez, Juan. "MPU9250PhotonLibrary." *GitHub*, 9 Oct. 2018, https://github.com/jdvr1994/MPU9250PhotonLibrary
[2] InvenSense, "MPU-9250 Register Map and Descriptions Revision 1.4," RM-MPU-9250A-00 datasheet, September 2013
[3] InvenSense, "MPU-9250 Product Specification Revision 1.1," PS-MPU-9250A-01 datasheet, June 2016
[4] Free Tutorials. "Understanding MPU6050 ACC full scale range," *YouTube,* Oct. 8, 2019 [Video file]. Available:

https://www.youtube.com/watch?v=e28SHRiJBQY. [Accessed: Feb. 28, 2020].
[5] Laukkonen, Jeremy. "Bluetooth vs. Wi-Fi: What's the Difference?," Lifewire,
Feb. 2020. Accessed on: Feb. 26, 2020. [Online]
[6] On, Amlendra. "Difference between I2C and SPI (I2C vs SPI), you should know.," Aticleworld, Accessed on: March 1, 2020. [Online]
[7] SFUPTOWNMAKER. "I2C at the Hardware Level," SparkFun, Accessed on: Feb. 28, 2020. [Online]
[8] AddOhms. "Picking Pull-Up Resistor Values | AO #25," *YouTube,* May. 3, 2018 [Video file]. Available: https://www.youtube.com/watch?v=u3Xiy2DVnI4. [Accessed: March 2, 2020].
[9] Hoffman, Chris. "What's the Difference Between TCP and UDP?," HowToGeek,
Jul. 2017. Accessed on: Feb. 28, 2020. [Online]
[10] Particle, "Photon Datasheet" V016 datasheet
[11] rocksetta. "TCP Server and Client Example Socket Programs "D7-ON" please," Particle, Accessed on: Feb. 28, 2020. [Online]
[12] Balzer82. "Kalman." *GitHub*, 12 March 2018, https://github.com/balzer82/Kalman
[13] kangalow. "Jetson Nano – Using I2C" Wordpress, Jul. 2019. Accessed on: Feb. 28, 2020. [Online]



Tasks:	24 Feb	2 Mar	9 Mar	16 Mar	23 Mar	30 Mar	6 Apr	13 Apr	20 Apr	27 Apr	4 May
Motion Sensing			🟨					🟨			
Analyze process and planning	DONE		🟨					🟨			
Ordering parts	DONE		🟨					🟨			
Reading data through I2C	🟩	🟩	🟨					🟨			
IMU data transmission (wifi)	🟥	🟥	🟨					🟨			
IMU-position mapping	🟦	🟦	🟨	🟪				🟨			
Dog Simulator			🟨					🟨			
Analyze process and planning	DONE		🟨					🟨			
Projectile prediction - physics			🟨	🟦				🟨			
Order parts for Dog	DONE		🟨					🟨			
IMU data transmission (wifi)	🟥		🟨					🟨			
Build mobile base for dog			🟨	🟥	🟩	🟥		🟨			
Controls mobile base (wheels and motors)			🟨	🟩	🟪	🟪	🟪	🟨			
Catching/retrieval			BREAK	🟪	🟥	🟦	🟪	🟨			
Returning the ball			🟨		🟦	🟩	🟪	🟨			
Integration & Testing			🟨					CARNIVAL	🟪	🟪	
Slack Time			🟨					🟨			⬛
	Key:	🟦	🟩	🟥	🟪						
		Daniel	Luca	Hana	All						