Robotics:

Coding Over the Summer

Contents

[The Schedule of Meetings 4](#_Toc517013315)

[Week 1 6/27/2018 4](#_Toc517013316)

[Week 2 7/11/2018 4](#_Toc517013317)

[Week 3 7/18/2018 4](#_Toc517013318)

[Week 4 7/25/2018 5](#_Toc517013319)

[Week 5 8/1/2018 5](#_Toc517013320)

[Week 6 8/8/2018 5](#_Toc517013321)

[Week 7 8/15/2018 6](#_Toc517013322)

[Week 8 8/22/2018 6](#_Toc517013323)

[Raspberry Pi 6](#_Toc517013324)

[Download the image 6](#_Toc517013325)

[Writing an image to the SD card 7](#_Toc517013326)

[Sparkfun Micro:Bot 7](#_Toc517013327)

[Encoders 10](#_Toc517013328)

[Eclipse 10](#_Toc517013329)

[Launch Pi 11](#_Toc517013330)

[**Use Launch Pi just like Run As Deploy for WPILIB** 11](#_Toc517013331)

[**Installation** 11](#_Toc517013332)

[**Code** 11](#_Toc517013333)

[**Pi set up** 12](#_Toc517013334)

[**LaunchPi configuration** 12](#_Toc517013335)

[WiringPi 12](#_Toc517013336)

[Download and Install 12](#_Toc517013337)

[Pi4j 14](#_Toc517013338)

[[Source: http://pi4j.com/install.html ] 14](#_Toc517013339)

[Installation 14](#_Toc517013340)

[Easy/Preferred 14](#_Toc517013341)

[Download Library for Java 14](#_Toc517013342)

[Pin Numbering - Raspberry Pi 2 Model B 15](#_Toc517013343)

[Numbering Scheme 15](#_Toc517013344)

[Expansion Header 16](#_Toc517013345)

[J8 Pinout (40-pin Header) 16](#_Toc517013346)

[Additional Resources 18](#_Toc517013347)

[JAMA 18](#_Toc517013348)

[Capabilities. 18](#_Toc517013349)

[Kalman Filter Basic Code 19](#_Toc517013350)

[NIST Implementation by Ahmed Abdelkader 19](#_Toc517013351)

[Kalman Filter used in FRC 20](#_Toc517013352)

[Linux Scripts 22](#_Toc517013353)

[Auto Start Java Code 22](#_Toc517013354)

[Jar-start.sh 22](#_Toc517013355)

[Jar-stop.sh 22](#_Toc517013356)

[Control script 23](#_Toc517013357)

[/etc/init.d/robotcode.sh 23](#_Toc517013358)

[Install script into startup procedures 23](#_Toc517013359)

[FRC Build Process 24](#_Toc517013360)

[Build.xml 24](#_Toc517013361)

[C:\Users\<userid>\wpilib\wpilib.propertie 24](#_Toc517013362)

[C:\Users\<userid>\wpilib\java\current\ant\build.properties 25](#_Toc517013363)

[Robot Code Build Process 27](#_Toc517013364)

[Build.xml 27](#_Toc517013365)

# The Schedule of Meetings

## Week 1 6/27/2018

The main points for this week are to get across the scope of the project and get the students set up with the programming environment and their Raspberry Pi.

* Eclipse
  + Jsch.jar
  + Pi4j.jar
  + WiringPi.jar
* Raspberry Pi
  + SSH
  + Pi4j.jar
  + WiringPi.jar
* Github
  + Individual accounts
  + Group account
* Ant
  + The Ant script

## Week 2 7/11/2018

The requirements of the system that we are building and how those requirements are going to be met. Converting those project requirements into design requirements. Looking at how the elements chosen for the robot will meet those requirements.

## Week 3 7/18/2018

Programming a system to drive the robot. The specific technical requirements of the TB6612FNG controller will drive how we run the system. There is probably some good sample code that we can implement for this. Our discussion point is how to integrate this code into our system.

|  |  |  |  |
| --- | --- | --- | --- |
| **Pin Label** | **Function** | **Power/Input/Output** | **Notes** |
| VM | Motor Voltage | Power | This is where you provide power for the motors (2.2V to 13.5V) |
| VCC | Logic Voltage | Power | This is the voltage to power the chip and talk to the microcontroller (2.7V to 5.5V) |
| GND | Ground | Power | Common Ground for both motor voltage and logic voltage (all GND pins are connected) |
| STBY | Standby | Input | Allows the H-bridges to work when high (has a pulldown resistor so it must actively pulled high) |
| AIN1/BIN1 | Input 1 for channels A/B | Input | One of the two inputs that determines the direction. |
| AIN2/BIN2 | Input 2 for channels A/B | Input | One of the two inputs that determines the direction. |
| PWMA/PWMB | PWM input for channels A/B | Input | PWM input that controls the speed |
| A01/B01 | Output 1 for channels A/B | Output | One of the two outputs to connect the motor |
| A02/B02 | Output 2 for channels A/B | Output | One of the two outputs to connect the motor |

## Week 4 7/25/2018

This week we are going to be talking about how to run a separate process. The DAGU encoders that we are using are merely going to change the state of the GPIO pin to which they are connected. We will discuss the implications of making measurements on the motor side of the system and work through all of the math about how this can help us navigate.

## Week 5 8/1/2018

This week we are going to talk about I2C and how to talk to the MPU-9250 Gyro and process its data. There are some decision that have to be made here that might influence our entire architecture. There is a set of Pi4J routines to handle I2C communication and there is also the processing of the gyro signals. The two make for quite a tough set of concepts to cover. If we punt anything here, it should be the I2C functions, which we will want to do in depth when we cover the Arduino interface.

## Week 6 8/8/2018

This week we cover the introduction of a second system to the equation. We discuss the different protocols, but stick with the I2C protocols, because we already have a leg up on the system. We can just take some stock code for the distance sensors and set up the registers to pass the data back to the Raspberry Pi. This will be of value to us later on when we look at the different ways of communicating with other vision systems. Our analog sensors this week are the Sharp GP2Y0A21YK0F sensors with a range of 10 cm to 80 cm.

## Week 7 8/15/2018

By week 7 we should be able to put everything we have together and discuss coordinate systems in detail. Instead of steering by Gyro that we did in Week 5, we can talk about putting the Gyro and Encoder data together and performing autonomous steering.

## Week 8 8/22/2018

We finish up the summer with an introduction to matrix operations and the Java Matrix library.

# Raspberry Pi

[Source: <https://www.raspberrypi.org/documentation/installation/installing-images/> ]

This resource explains how to install a Raspberry Pi operating system image on an SD card. You will need another computer with an SD card reader to install the image.

We recommend most users download [NOOBS](https://www.raspberrypi.org/documentation/installation/noobs.md), which is designed to be very easy to use. However, more advanced users looking to install a particular image should use this guide.

## Download the image

Official images for recommended operating systems are available to download from the Raspberry Pi website [Downloads page](https://www.raspberrypi.org/downloads/).

Alternative distributions are available from third-party vendors.

If you're not using Etcher (see below), you'll need to unzip .zip downloads to get the image file (.img) to write to your SD card.

**Note**: the Raspbian with Raspberry Pi Desktop image contained in the ZIP archive is over 4GB in size and uses the [ZIP64](https://en.wikipedia.org/wiki/Zip_(file_format)#ZIP64) format. To uncompress the archive, a unzip tool that supports ZIP64 is required. The following zip tools support ZIP64:

* [7-Zip](http://www.7-zip.org/) (Windows)
* [The Unarchiver](http://unarchiver.c3.cx/unarchiver) (Mac)
* [Unzip](http://www.info-zip.org/mans/unzip.html) (Linux)

## Writing an image to the SD card

You will need to use an image writing tool to install the image you have downloaded on your SD card.

**Etcher** is a graphical SD card writing tool that works on Mac OS, Linux and Windows, and is the easiest option for most users. Etcher also supports writing images directly from the zip file, without any unzipping required. To write your image with Etcher:

* Download [Etcher](https://etcher.io/) and install it.
* Connect an SD card reader with the SD card inside.
* Open Etcher and select from your hard drive the Raspberry Pi .img or .zip file you wish to write to the SD card.
* Select the SD card you wish to write your image to.
* Review your selections and click 'Flash!' to begin writing data to the SD card.

For more advanced control of this process, see our system-specific guides:

* [Linux](https://www.raspberrypi.org/documentation/installation/installing-images/linux.md)
* [Mac OS](https://www.raspberrypi.org/documentation/installation/installing-images/mac.md)
* [Windows](https://www.raspberrypi.org/documentation/installation/installing-images/windows.md)

If you already have a Raspberry Pi and use an older image, you will need to get a Java compiler. New installations come with the JDK, you can install one using:

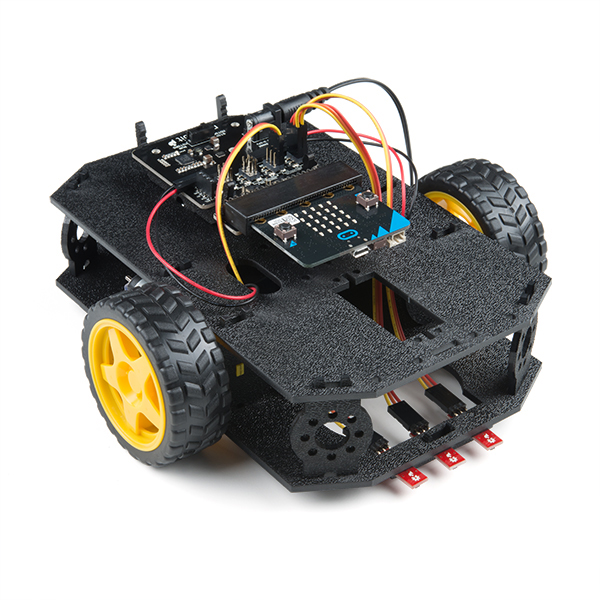
sudo apt-get update && sudo apt-get install oracle-java7-jdk

# Sparkfun Micro:Bot

[Source: <https://www.sparkfun.com/products/14216> ]



* 1x [SparkFun moto:bit](https://www.sparkfun.com/products/14213)
* 1x [Shadow Chassis](https://www.sparkfun.com/products/13301)
* 3x [SparkFun RedBot Sensor — Line Follower](https://www.sparkfun.com/products/11769)
* 2x [Sub-Micro Servo](https://www.sparkfun.com/products/9065)
* 2x [Hobby Gearmotor](https://www.sparkfun.com/products/13302)
* 2x [Wheel — 65mm (Rubber Tire, Pair)](https://www.sparkfun.com/products/13259)
* 3x [Jumper Wire — 3-pin, 6"](https://www.sparkfun.com/products/10368)



# Encoders

[Source: <https://www.kr4.us/Wheel-Encoder-Kit.html?gclid=CjwKCAjw9e3YBRBcEiwAzjCJukiQcxWzFPZFt0HymlmpqW3kweqYVTNdE9u5tmmqom47ZPbYAewLzBoCdP8QAvD_BwE> ]

These encoders consist of a magnetic sensor and a disk with 8 magnets embedded. The driver has to count the number of times the magnets go past the sensors. The motor speed is 90 revolutions per minute, which will drive the wheels on a 48:1 gearbox. The disks will spin about 1.4 times per second and give a total of 16 signals, one rising and one falling, per revolution. The programming task will be to capture every one of these transitions and increment the count so we have an accurate count of how many times the wheels go around.

# Eclipse

We need to have the Java Development Kit installed first.

<https://docs.oracle.com/javase/9/install/installation-jdk-and-jre-microsoft-windows-platforms.htm>

Install Eclipse if you do not have it. Make sure it is the same version that your team will be using.

<https://www.eclipse.org/downloads/eclipse-packages/?show_instructions=TRUE>

# Launch Pi

**Use Launch Pi just like Run As Deploy for WPILIB**

[FROM: <https://robotjava.wordpress.com/2016/10/06/deploying-java-to-raspberry-pi-remotely-using-launch-pi-and-eclipse/> ]

**Installation**

1. Install the LaunchPi plugin by going to Help -> Install New Software. Click Add.
   * Name: Launch Pi
   * Location: <https://raw.githubusercontent.com/tsvetan-stoyanov/launchpi/master/org.launchpi.us/target/site/site.xml>
2. Select all and finish. Restart Eclipse
3. File -> New -> Other. Search or find Gradle Project. Enter your project name and press Finish
4. In the new project, open build.gradle and add the following in the dependencies section
5. dependencies {
6. compile group: 'com.pi4j', name: 'pi4j-core', version: '1.1'
8. ...
9. }
10. Right click the project, Gradle -> Refresh Gradle Project

**Code**

We will test the deploy by making a light blink. Create a class called TestBlinkingLights.java and paste the below then build.

import com.pi4j.io.gpio.\*;

public class TestBlinkingLights {

public static void main(String[] args) throws InterruptedException {

final GpioController gpio = GpioFactory.getInstance();

final GpioPinDigitalOutput outputPIN = gpio.provisionDigitalOutputPin(RaspiPin.GPIO\_08, "OutputPIN", PinState.LOW);

outputPIN.setShutdownOptions(true);

while (true) {

Thread.sleep(1000);

outputPIN.high();

Thread.sleep(1000);

outputPIN.low();

}

}

}

**Pi set up**

As you can see, we will use GPIO 8 on Pi4J pin setup, which is pin 3 on the Pi. I used Pin 9 as the Ground. Refer to my [LED post](https://couchwarriorblog.wordpress.com/2016/08/29/let-there-be-led/) for how to set up the bread board. In our case, the “power” is now coming from pin 3, not a 3.3V pin.

**LaunchPi configuration**

In order to deploy the pi, your desktop and the Pi must be on the same network. You also need the Pi’s ip address. Different methods [here](https://www.raspberrypi.org/documentation/remote-access/ip-address.md). I just issued command “hostname -I” on the Pi’s terminal. Back in Eclipse, create a new Run configuration

1. Run -> Run Configurations…
2. Right-click on RaspberryPi in the left hand side and choose New
3. Raspberry PI tab
   * Click New and enter your Pi’s ip address. Leave everything else alone.
   * Debug port: 4000
   * X Display: 0
   * Run java as root: checked
4. Main tab
   * Project: Browse for your project
   * Main class: Search for your the TestBlinkingLights class
   * Leave other fields blank
5. Click Apply and Run
6. Your Pi’s default user name is pi
7. Your Pi’s default password is raspberry
8. The LED should now blink

# WiringPi

[FROM: <http://wiringpi.com/download-and-install/> ]

# Download and Install

***WiringPi*** is maintained under GIT for ease of change tracking, however there is a Plan B if you’re unable to use GIT for whatever reasons (usually your firewall will be blocking you, so do check that first!)

***Note***: wiringPi is **NOT** hosted on Github. There are many forks that you may find there, but they are not the original version maintained by myself.

To view the wiringPi sources, then go to:

<https://git.drogon.net/>

and select the wiringPi link.

#### To install…

First check that wiringPi is not already installed. In a terminal, run:

$ gpio -v

If you get something, then you have it already installed. The next step is to work out if it’s installed via a standard package or from source. If you installed it from source, then you know what you’re doing – carry on – but if it’s installed as a package, you will need to remove the package first. To do this:

$ sudo apt-get purge wiringpi

$ hash -r

Then carry on.

If you do not have GIT installed, then under any of the Debian releases (e.g. Raspbian), you can install it with:

$ sudo apt-get install git-core

If you get any errors here, make sure your Pi is up to date with the latest versions of Raspbian: (this is a good idea to do regularly, anyway)

$ sudo apt-get update

$ sudo apt-get upgrade

To obtain WiringPi using GIT:

$ cd

$ git clone git://git.drogon.net/wiringPi

If you have already used the clone operation for the first time, then

$ cd ~/wiringPi

$ git pull origin

Will fetch an updated version then you can re-run the build script below.

To build/install there is a new simplified script:

$ cd ~/wiringPi

$ ./build

The new build script will compile and install it all for you – it does use the sudo command at one point, so you may wish to inspect the script before running it.

# Pi4j

### [Source: <http://pi4j.com/install.html> ]

## Installation

### Easy/Preferred

(NOTE: This installation method requires that your RaspberryPi is connected to the Internet.)

The simplest method to install Pi4J on your RaspberryPi is to execute the following command directly on your RaspberryPi.  
curl -s get.pi4j.com | sudo bash

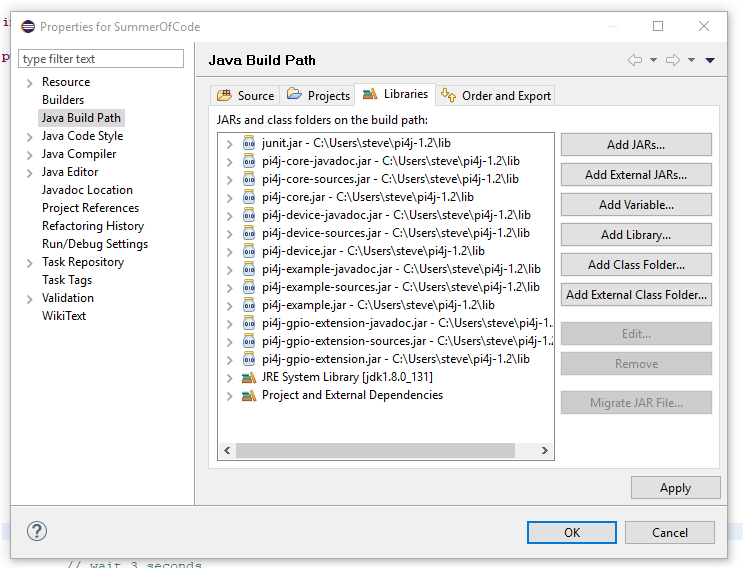
This method will download and launch an installation script that perform the following steps:

1. adds the Pi4J APT repository to the local APT repositories
2. downloads and installs the Pi4J GPG public key for signature validation
3. invokes the 'apt-get update' command on the Pi4J APT repository to update the local package database
4. invokes the 'apt-get install pi4j' command to perform the download and installation

## Download Library for Java

[Source: <http://pi4j.com/download.html> ]

Once you have eclipse loaded you have to add the libraries to the Build. This makes the functions available to your code. Use the Jar files for SNAPSHOT 1.2.



## Pin Numbering - Raspberry Pi 2 Model B

* [Numbering Scheme](http://pi4j.com/pins/model-2b-rev1.html#Numbering_Scheme)
* [Expansion Header](http://pi4j.com/pins/model-2b-rev1.html#Expansion_Header)
* [J8 Pinout (40-pin Header)](http://pi4j.com/pins/model-2b-rev1.html#J8_Pinout_40-pin_Header)
* [Additional Resources](http://pi4j.com/pins/model-2b-rev1.html#Additional_Resources)

### Numbering Scheme

Pi4J (by default) uses an abstract pin numbering scheme to help insulate software from hardware changes.  
Pi4J implements the same pin number scheme as the Wiring Pi project. More information about the WiringPi pin number scheme can be found here: <http://wiringpi.com/pins/>

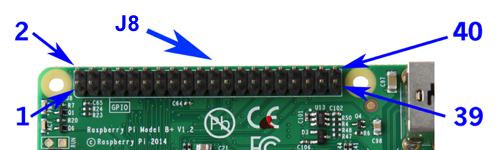
Pi4J provides a [RaspiPin](http://pi4j.com/apidocs/index.html?com/pi4j/io/gpio/RaspiPin.html) enumeration that is used to manage the accessible GPIO pins.

(NOTE: Pi4J also can be configured to use the Broadcom Pin numbering scheme.)

Please see this page for more information on both the WiringPi and Broadcom pin numbering schemes:   
>> [Pin Numbering Schemes](http://pi4j.com/pin-numbering-scheme.html)

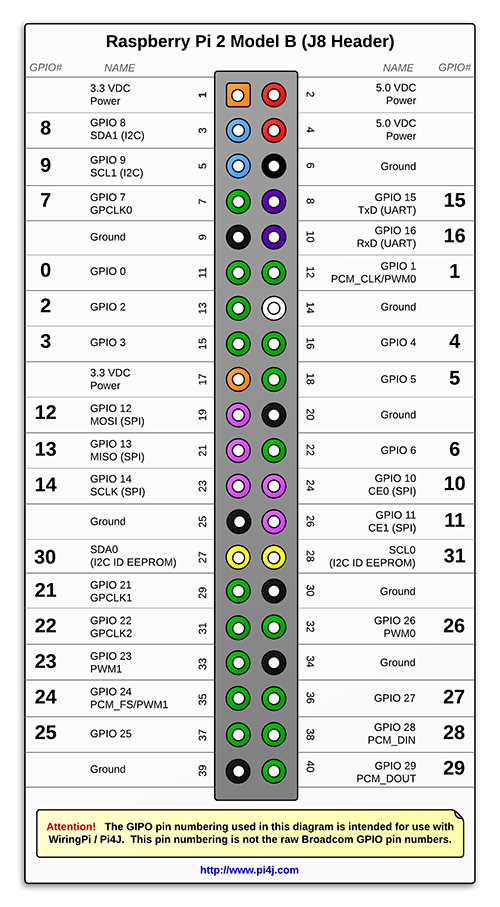
### Expansion Header

The Raspberry Pi 2 Model B board contains a single 40-pin expansion header labeled as 'J8' providing access to 28 GPIO pins.  
(Pins 1, 2, 39 & 40 are also labeled below.)



### J8 Pinout (40-pin Header)

The diagram below illustrates the GPIO pinout using the Pi4J/WiringPi GPIO numbering scheme.



[(click here for hi-resolution image)](http://pi4j.com/images/j8header-2b-large.png)

### Additional Resources

* Please visit the [usage](http://pi4j.com/usage.html) page for additional details on how to control these pins using Pi4J.
* [Click here for more information on the J8 header.](http://www.raspberrypi.org/products/raspberry-pi-2-model-b/)
* [Click here for more information the Raspberry Pi pin functions.](http://elinux.org/RPi_BCM2835_GPIOs)

# JAMA

[Source: <https://math.nist.gov/javanumerics/jama/> ]

JAMA is a basic linear algebra package for Java. It provides user-level classes for constructing and manipulating real, dense matrices. It is meant to provide sufficient functionality for routine problems, packaged in a way that is natural and understandable to non-experts. It is intended to serve as the standard matrix class for Java, and will be proposed as such to the [Java Grande Forum](http://www.nist.gov/cgi-bin/exit_nist.cgi?timeout=5&url=http://www.npac.syr.edu/javagrande/) and then to [Sun](http://www.nist.gov/cgi-bin/exit_nist.cgi?timeout=5&url=http://java.sun.com/). A straightforward public-domain reference implementation has been developed by the [MathWorks](http://www.mathworks.com/) and [NIST](http://www.nist.gov/) as a strawman for such a class. We are releasing this version in order to obtain public comment. There is no guarantee that future versions of JAMA will be compatible with this one.

A sibling matrix package, [Jampack](ftp://math.nist.gov/pub/Jampack/Jampack/AboutJampack.html), has also been developed at NIST and the University of Maryland. The two packages arose from the need to evaluate alternate designs for the implementation of matrices in Java. JAMA is based on a single matrix class within a strictly object-oriented framework. Jampack uses a more open approach that lends itself to extension by the user. As it turns out, for the casual user the packages differ principally in the syntax of the matrix operations. We hope you will take the time to look at Jampack along with JAMA. There is much to be learned from both packages.

### Capabilities.

JAMA is comprised of six Java classes: Matrix, CholeskyDecomposition, LUDecomposition, QRDecomposition, SingularValueDecomposition and EigenvalueDecomposition.

The Matrix class provides the fundamental operations of numerical linear algebra. Various constructors create Matrices from two dimensional arrays of double precision floating point numbers. Various *gets* and *sets* provide access to submatrices and matrix elements. The basic arithmetic operations include matrix addition and multiplication, matrix norms and selected element-by-element array operations. A convenient matrix print method is also included.

Five fundamental matrix decompositions, which consist of pairs or triples of matrices, permutation vectors, and the like, produce results in five decomposition classes. These decompositions are accessed by the Matrix class to compute solutions of simultaneous linear equations, determinants, inverses and other matrix functions. The five decompositions are

* Cholesky Decomposition of symmetric, positive definite matrices
* LU Decomposition (Gaussian elimination) of rectangular matrices
* QR Decomposition of rectangular matrices
* Eigenvalue Decomposition of both symmetric and nonsymmetric square matrices
* Singular Value Decomposition of rectangular matrices

The current JAMA deals only with real matrices. We expect that future versions will also address complex matrices. This has been deferred since crucial design decisions cannot be made until certain issues regarding the implementation of complex in the Java language are resolved.

# Kalman Filter Basic Code

## NIST Implementation by Ahmed Abdelkader

import Jama.Matrix;

/\*\*

\* This work is licensed under a Creative Commons Attribution 3.0 License.

\* @author Ahmed Abdelkader

\*/

public class KalmanFilter {

protected Matrix X, X0;

protected Matrix F, B, U, Q;

protected Matrix H, R;

protected Matrix P, P0;

public void predict() {

X0 = F.times(X).plus(B.times(U));

P0 = F.times(P).times(F.transpose()).plus(Q);

}

public void correct(Matrix Z) {

Matrix S = H.times(P0).times(H.transpose()).plus(R);

Matrix K = P0.times(H.transpose()).times(S.inverse());

X = X0.plus(K.times(Z.minus(H.times(X0))));

Matrix I = Matrix.identity(P0.getRowDimension(), P0.getColumnDimension());

P = (I.minus(K.times(H))).times(P0);

}

//getters and setters go here

}

## Kalman Filter used in FRC

package org.usfirst.frc.team1294.robot.filter;

/\*\*

\* {@link SimpleKalman} is an one dimensional implementation of KALMAN filter used to

\* predict a noisy signal containing fluctuations due to measurement errors, noise etc.

\*

\* @author AchuthaRanga.Chenna https://gist.github.com/achutharanga/4e542c39bcbd28071d4f

\*

\*/

public class SimpleKalman {

/\*\* process noise COVARIANCE \*/

private final double q;

/\*\* measurement noise COVARIANCE \*/

private final double r;

/\*\* Measurement value \*/

private double x = 0d;

/\*\* estimation error COVARIANCE \*/

private double p = 50d;

/\*\* KALMAN gain \*/

private double k = 0.04d;

/\*\*

\* Init the filter with initial value of signal.

\* @param initial\_measurement

\*/

public SimpleKalman(double q, double r, double initial\_measurement) {

this.q = q;

this.r = r;

this.x = initial\_measurement;

}

/\*\*

\* Method to get the predicted value of the measured signal value.

\*

\* @param measurement

\* value from real time data.

\* @return predicted measurement.

\*/

public double getPredicted\_Value(double measurement) {

p = p + q;

k = p / (p + r);

x = x + k \* (measurement - x);

p = (1 - k) \* p;

return x;

}

}

# Linux Scripts

In order to get your java code to start up when the Raspberry Pi starts, you have to get it into the startup sequence. In Linux, this is a set of scripts running out of /etc/init.d and called during startup. We need a script to start the jar file and one to stop the execution of the jar file.

## Auto Start Java Code

Start Script: /usr/local/bin/jar-start.sh

Stop Script: /usr/local/bin/jar-stop.sh

### Jar-start.sh

#!/bin/bash

java -jar /home/pi/robot/robotcode.jar

### Jar-stop.sh

#!/bin/bash

pid=`ps aux | grep robotcode | awk '{print $2}'`

kill -9 $pid

## Control script

### /etc/init.d/robotcode.sh

#!/bin/bash

# MyApp

#

# description: your app description

case $1 in

start)

/bin/bash /usr/local/bin/jar-start.sh

;;

stop)

/bin/bash /usr/local/bin/jar-stop.sh

;;

restart)

/bin/bash /usr/local/bin/jar-stop.sh

/bin/bash /usr/local/bin/jar-start.sh

;;

esac

exit 0

### Install script into startup procedures

update-rc.d robotcode defaults

# FRC Build Process

This is an ant build process. Ant gets is instructions from build.xml, which it normally finds in the same directory from which ant is run. It expects ant and javac to be on the system path. FRC places a number of parameters in separate files. Notice that the first file “wpilib.properties” provides the definition for the variable “${version} , which provides the path to the third file.

## Build.xml

<?xml version="1.0" encoding="UTF-8"?>

<project name="FRC Deployment" default="deploy">

<!--

The following properties can be defined to override system level

settings. These should not be touched unless you know what you're

doing. The primary use is to override the wpilib version when

working with older robots that can't compile with the latest

libraries.

-->

<!-- By default the system version of WPI is used -->

<!-- <property name="version" value=""/> -->

<!-- By default the system team number is used -->

<!-- <property name="team-number" value=""/> -->

<!-- By default the target is set to 10.TE.AM.2 -->

<!-- <property name="target" value=""/> -->

<!-- Any other property in build.properties can also be overridden. -->

<property file="${user.home}/wpilib/wpilib.properties"/>

<property file="build.properties"/>

<property file="${user.home}/wpilib/java/${version}/ant/build.properties"/>

<import file="${wpilib.ant.dir}/build.xml"/>

</project>

## C:\Users\<userid>\wpilib\wpilib.properties

#Don't add new properties, they will be deleted by the eclipse plugin.

#Fri Jun 08 20:44:43 EDT 2018

version=current

team-number=4949

## C:\Users\<userid>\wpilib\java\current\ant\build.properties

# Deployment information

username=lvuser

password=

adminUsername=admin

adminPassword=

deploy.dir=/home/lvuser

libDeploy.dir=/usr/local/frc/lib

deploy.kill.command=. /etc/profile.d/natinst-path.sh; /usr/local/frc/bin/frcKillRobot.sh -t -r

debug.flag.dir=/tmp/

debug.flag.command=chown lvuser:ni ${debug.flag.dir}frcdebug

command.dir=/home/lvuser/

version=current

roboRIOJRE.dir=/usr/local/frc/JRE

roboRIOJRE.ipk=zulu-jre\_1.8.0-131\_cortexa9-vfpv3.ipk

# Libraries to use

wpilib=${user.home}/wpilib/java/${version}

wpilib.common=${user.home}/wpilib/common/${version}

wpilib.lib=${wpilib}/lib

wpilib.native.lib=${wpilib.common}/lib/linux/athena/shared

wpilib.jar=${wpilib.lib}/WPILib.jar

wpilib.sources=${wpilib.lib}/WPILib-sources.jar

ntcore.jar=${wpilib.lib}/ntcore.jar

ntcore.sources=${wpilib.lib}/ntcore-sources.jar

wpiutil.jar=${wpilib.lib}/wpiutil.jar

wpiutil.sources=${wpilib.lib}/wpiutil-sources.jar

opencv.jar=${wpilib.lib}/opencv.jar

opencv.sources=${wpilib.lib}/opencv-sources.jar

cscore.jar=${wpilib.lib}/cscore.jar

cscore.sources=${wpilib.lib}/cscore-sources.jar

classpath=${wpilib.jar}:${ntcore.jar}

userLibs.dir=${user.home}/wpilib/user/java/lib

# Ant support

wpilib.ant.dir=${wpilib}/ant

jsch.jar=${wpilib.ant.dir}/jsch-0.1.50.jar

classloadertask.jar=${wpilib.ant.dir}/ant-classloadertask.jar

# Build information

jar=FRCUserProgram.jar

src.dir=src

build.dir=build

build.jars=${build.dir}/jars

dist.dir=dist

dist.jar=${dist.dir}/${jar}

# Simulation Information

simulation.dist.jar=${dist.dir}/FRCUserProgramSim.jar

wpilib.sim=${user.home}/wpilib/simulation

wpilib.sim.lib=${wpilib.sim}/jar

wpilib.sim.tools=${wpilib.sim}/tools

# Robot Code Build Process

## Build.xml

<?xml version="1.0" encoding="UTF-8"?>

<!-- ======================================================================

Jun 10, 2018 2:41:21 PM

Summer Of Code

steve

====================================================================== -->

<project name="Compile and Run Summer of Code" default="deploy">

<description>

This is an example build file that can set up the build structure, compile

a jar file, and deploy it to the Raspberry Pi.

</description>

<!-- set global properties for this build -->

<property name="src" location="src"/>

<property name="build" location="build"/>

<property name="dist" location="dist"/>

<!-- =================================

target: init

================================= -->

<target name="init">

<!-- Create the build directory structure used by compile -->

<mkdir dir="${build}"/>

</target>

<!-- =================================

target: compile

================================= -->

<target name="compile" depends="init">

<javac srcdir="${src}" destdir="${build}" >

<classpath>

<pathelement path="C:\Users\steve\ntcore-java\ntcore-java-4.0.0.jar"/>

<pathelement path="C:\Users\steve\ntcore-java\ntcore-jni-4.0.0-all.jar"/>

<pathelement path="C:\Users\steve\ntcore-java\wpiutil-java-3.0.0.jar"/>

<pathelement path="C:\Users\steve\pi4j-1.2\lib\pi4j-core.jar"/>

<pathelement path="C:\Users\steve\pi4j-1.2\lib\pi4j-device.jar"/>

<pathelement path="C:\Users\steve\pi4j-1.2\lib\pi4j-gpio-extension.jar"/>

</classpath>

</javac>

</target>

<!-- - - - - - - - - - - - - - - - - -

target: dist

- - - - - - - - - - - - - - - - - -->

<target name="dist" depends="compile"

description="generate the distribution">

<!-- Create the distribution directory -->

<mkdir dir="${dist}/lib"/>

<!-- Put everything in ${build} into the PHSUserProgram.jar file -->

<jar jarfile="${dist}/lib/PHSUserProgram.jar" basedir="${build}"/>

</target>

<!-- - - - - - - - - - - - - - - - - -

target: deploy

- - - - - - - - - - - - - - - - - -->

<target name="deploy">

<scp todir="pi:raspb3rry@172.16.0.228:/home/pi" file="${dist}/lib/PHSUserProgram.jar"></scp>

</target>

<!-- - - - - - - - - - - - - - - - - -

target: run

- - - - - - - - - - - - - - - - - -->

<target name="run" depends = "dist">

</target>

<!-- - - - - - - - - - - - - - - - - -

target: clean

- - - - - - - - - - - - - - - - - -->

<target name="clean"

description="clean up">

<!-- Delete the ${build} and ${dist} directory trees -->

<delete dir="${build}"/>

<delete dir="${dist}"/>

</target>

</project>