**Caffeinator - Software API Update**

Document Version Number: 3.0

Date: April 12th, 2015

Author: Adrian Pan, Eric Vuong, James Fuh

Software: Final code

Goal: Application Programming Interface for Caffeinator

Table of Contents

Class Odometer 1

Class Navigator 3

Class Robot extends Class Navigator 4

Class USLocalizer 4

Class LightLocalizer 5

Class DifferentialFiltering 5

Class OdometryDisplay 5

Class Shoot 6

Class shootingCoordinate\* 6

# Class Odometer

**void run()**   
Robot’s odometer will constantly update its position in X and Y (cm) as well as its theta (rad).

**Double stickToSameCircle(double rads)**  
This method takes input a theta value in radian and will convert it to a value between 0 and 2pi.

**void getPosition(double [] position, boolean [] update)**This method will pass the value of X, Y and theta to the input array: position[]. It has an accessor function. Note: the values can only be passed if the corresponding boolean array is set to true.

**void getPosition(double [] pos)**This method will pass the value of X, Y and theta to the input array: position[]. It has an accessor function. There is no boolean condition.

**double getX()**   
Return the odometer’s X position.

**double getY()**   
Return the odometer’s Y position.

**double getTheta()**   
Return the odometer’s theta in radian.

**double getAngle()**   
Return the odometer’s theta in degree.

**void setPosition(double [] position, boolean [] update)**This method will set the value of X, Y and theta by the input array: position[]. It has a setter function. Note: the values can only be set if the corresponding boolean array is set to true.

**void setX(double x)**   
Set the odometer’s X position.

**void setY(double y)**   
Set the odometer’s Y position.

**void setTheta(double theta)**   
Set the odometer’s theta in radian.

# Class Navigator

**void run()**

The robot will pause for a few milliseconds and run the navigator mode.

**void travelTo(double x, double y)**   
Robot will drive to desired destination. X and Y are entered in cm.

**void setSpeed(int speed)**  
Set both speed of the motor. Sets desired motor speed , in degrees per second; The maximum reliably sustainable velocity is 100 x battery voltage under moderate load, such as a direct drive robot on the level.

**void turnTo(double theta)**Robot will turn to desired angle in absolute coordinate where 0 degrees is + x-axis, increasing counterclockwise. Theta is entered in radians. Turns to minimal angle.

**void turn(double angle)**  
Turn a certain degree relative to robot’s current orientation. Angle is entered in degrees.

**void** **driveTo(double** **x, double** **y)**

The robot will turn and drive to the specific x and y coordinates in cm. Unlike travelTo the robot will not stop once the robot reaches the x and y coordinates.

**void driveDist(double distance)**   
Robot will drive a certain distance forward. Distance is entered in cm.

**boolean isNavigating()**   
Return true if robot is traveling to a waypoint.

**int** **convertAngle(double** **radius, double** **width, double** **angle)**

Convert the angle from radian to degree.

**int** **convertDistance(double** **radius, double** **distance)**

Calculate the distance in cm that the robot needs to travel

.

**double wrapAngle(double rads)**  
Wraps any given angle so it stays between 0 and 2pi (in radians).

**void sleep(int time)**  
Sleeps robot. Time is entered in milliseconds.

**int frontDist()**  
Returns the current front Ultrasonic Sensor reading in cm.

**int LeftDist()**   
Returns the current left Ultrasonic Sensor reading in cm.

**int RightDist()**   
Returns the current right Ultrasonic Sensor reading in cm.

**void** **doPAvoidanceLeft(double** **x, double** **y)**

Allow the robot to enter avoidance mode. The robot will avoid the obstacle with P- controller algorithm while it navigates to the destination. The robot utilizes the left sensor to keep track of the wall.

**void** **doPAvoidanceRight(double** **x, double** **y)**

Allow the robot to enter avoidance mode. The robot will avoid the obstacle with P- controller algorithm while it navigates to the destination. The robot utilizes the right sensor to keep track of the wall.

## Class Robot extends Class Navigator

**void setRotationSpeed(double speed)**Set the speed at which the robot rotates, in degrees per second.

**void setSpeed(double forwardSpeed, double rotationalSpeed)**Set the forward speed of the robot when it moves forward, and the rotational speed of the robot when it rotates, all in degrees per second.

# Class USLocalizer

**void doLocalization()**   
This method focuses on finding the minimum distance to determine location. The robot will first start turning counter-clockwise and the ultrasonic sensor will poll data once every 50ms. When two minimums are detected, the robot will stop and set the X, Y and theta accordingly.

**void setFilterValue()**set filter value for the ultrasonic sensor

**int getFilteredData()**   
The ultrasonic sensor does a ping and wait for 100ms for the ping to return. If the distance value is greater than the filter value, the distance value will be equal to filter value.

# Class LightLocalizer

**void doLocalization()**First the robot will determine the closest grid line intersection using odometer X and Y. Then it will use Navigator to travel to the starting position relative to the intersection and turn to starting orientation. The robot then starts to rotate counter-clockwise. When the light sensor detects each line, it will record its angle. After four lines are detected, the robot will stop turning and do calculation to determine its location. Afterward, the robot will travel to the closest intersection and turn to 0 degree.

# Class DifferentialFiltering

**Boolean lineDetection()**The light sensor will take a reading and save the value. After 50ms, it will take another reading and compare it with the initial reading. If their difference is larger than 50, it means a line is present and the method will return true. Otherwise, the previous reading will be equal to the current reading and the light sensor will wait 50ms again to get another reading.

# Class OdometryDisplay

**void run()**The display refreshes every 250ms. During this time, it will display X, Y, angle, front ultrasonic data and side ultrasonic data.

**String formattedDoubleToString(double x, int places)**

Convert double to string and return the string

# Class Shoot

**void launch()**   
One of the motors will rotate 360º, which will give enough force to hit the Ping-Pong ball.

# Class shootingCoordinate\*

\*This class is in a separate java project. It doesn’t require Lejos to run

**void main(String[] args)**   
The method will scan for all the possible coordinates for the launcher to hit the target and store them into an array list. After that, it will find the middle value of the array list and calculate the angle that the launcher needs to face. It will then print out the XY coordinate and angle in degrees. If there is no possible coordinate, it will display “no possible coordinate”.

Glossary of Terms

THETA – is always defined in radians

ANGLE – is always defined in degrees.