**API Caffeinator**

Table of Contents

[Class Odometer 1](#_Toc414806239)

[Class OdometryCorrection 2](#_Toc414806240)

[Class Navigator 2](#_Toc414806241)

[Class Robot extends Class Navigator 3](#_Toc414806242)

[Class USLocalizer 3](#_Toc414806243)

[Class LightLocalizer 4](#_Toc414806244)

[Class DifferentialFiltering 4](#_Toc414806245)

[Class OdometryDisplay 4](#_Toc414806246)

[Class Shoot 4](#_Toc414806247)

# 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 [] position)**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 setPosition(double [] position)**This method will set the value of X, Y and theta by the input array: position[]. It has a setter function. There is no boolean condition.

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

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

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

# Class OdometryCorrection

**void run()**   
This class is always running. When the light sensor passes by a grid line, robot’s odometer will correct its position in X and Y (cm) as well as its theta (rad),

# Class Navigator

**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.

**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.

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

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

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

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

**int sideDist()**   
Returns the current side Ultrasonic Sensor reading in cm.

## 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()**   
Choose between the new localization method or the old localization method

**void doAdvancedLocalization()**  
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 100ms. When two minimums are detected, the robot will stop and set the X, Y and theta accordingly.

**void doOrientation()**Robot will use the rising edge method to find its orientation

**void doCoordinate()**   
After the robot finds its orientation, it will first turn to face the wall at –Y and get distance. Then it turns to face the wall at –X and get distance. Afterward, it sets the odometer X and Y.

**int getFilteredData()**   
The ultrasonic sensor does a ping every 100ms. 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()**

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.