Yo//o API Description of important methods

Navigation.java

travelTo(double x, double y)

This is a method which takes a destination coordinates (x,y), in centimeters, and travels to said destination. Note that the destination is absolute and relies on the robot's current positioning according to the odometer.

1. Takes readings for the odometer to determine current robot location and heading
2. Determines angle to rotate in order for the robot to face the destination coordinates using the following equation: $angle_{dest} = \frac{180}{\pi} Math.a\tan 2(x_{dest} - x_{current}, y_{dest} - y_{current})$ where *Math.atan2* is the java function which calculates the phase (in radians) of given x-y coordinates from (0,0) (hence the subtraction of current coordinates from destination)
3. Calls turnTo() with the calculated heading to get the robot facing its destination. (see below for description of turnTo())
4. A while loop is then opened until destination is reached, which contains the following
   4.1. A constant update of current position and heading from odometry
   4.2. A calculation of error in current heading against expected heading (using the same equation as step 2, which allows for continuous angle correction
   4.3. Robot is then set to travel straight or to turn depending on whether an error in the heading exists, using the *TwoWheeledRobot.setRotationSpeed(angleError)* method (described below).

travelTo(double desiredAngle)

This is a method which rotates the robot to the angle/heading given, in degrees, absolute to North (0˚).

1. Stops both motors if a rotate function is currently being executed
2. Polls the odometer for current heading of the robot
3. Calculates the amount to rotate, in degrees, from the current heading to the desired heading
4. Calculates the amount of degrees of rotation needed in the wheels in order to rotate the amount determined in step 3 using the following equation:
   $$rotateAmount = \Delta\theta * \frac{width_{robot}}{2 \cdot radius_{wheel}}$$
5. Uses the *NXTRegulatedMotor.rotate()* method to rotate the left motor by -*rotateAmount* and the right motor by *rotateAmount*

USLocalizer.java

doLocalization()

This method allows for basic localization using one ultrasonic sensor (hereby referred to as *US*). This code is capable of both a rising and falling-edge type localization, but since only Rising edge is used in the main Task Scheduler, this localization technique will be outlined in the steps below

1. Rotate the robot using *TwoWheeledRobot.setSpeeds()* until a wall is detected (according to the set clipping value)
2. Keep rotating the robot until the wall is not detected anymore, and record the angle at which this happens (by polling the odometer). Note: headings recorded is relative to the robot's

heading at the start of the procedure, not North since no localization has been performed). Accuracy for this was increased by using two margins for considering a wall as ‘not seen’ and getting the average of the two

3. Rotation gets reversed, and step 2 is repeated, for the other side of the wall
4. The average of the two angles is considered to be an angle of approximately 225˚ from north, and the odometer is corrected accordingly.
5. Position is then estimated using *doInitialPosCalibration()* (see below)

doInitialPosCalibration()

This method uses *US* in order to estimate the robot’s position relative to the predefined (0,0) origin, as defined in the project specifications.

1. Rotate the robot to 270˚ (perpendicular to the wall) using the *Navigation.turnTo()* method, and log distance from the wall as reported by *US*
2. Update *x* position by subtracting 20 from the distance reported by the sensor (to account for offset of origin from corner of wall)
3. Repeat step 1 and 2 with a heading of 180˚ (perpendicular to the other wall), updating *y* position instead

LightLocalizer.java

doLocalization()

This method localizes the robot using a light sensor (hereby referred to as *LS*) oriented at the floor in order to provide more accurate localization than the preliminary one obtained by USLocalizer. Note: this assumes the localization starts in the (-x,-y) quadrant, relative to (0,0) (i.e facing southwest)

1. Poll odometer for current position and heading, and robot is set to rotate clockwise
2. If a gridline is detected, its heading is stored in an array
3. Steps 1 and 2 are repeated until the heading for the 4 gridlines (leaving the origin) are stored. We can therefore assume that the first and 3rd gridlines detected represent the negative and positive x-axis respectively, and the 2nd and 4th gridline represent the positive and negative y-axis, respectively.
4. The current position of the robot is then determined using the following equations:

$$pos_x = -dist_{LS}\cos\left(\theta_{y^-} - \theta_{y^+}\right)$$

$$pos_y = dist_{LS}\cos\left(\theta_{x^+} - \theta_{x^-}\right)$$

$$pos = (pos_x, pos_y)$$

$$\Delta\theta = \frac{\theta_{y^-} - \theta_{y^+}}{2} - \theta_{y^-}$$

$$\theta_{corrected} = \theta_{current} + \Delta\theta$$

where $dist_{LS}$ is the distance of *LS* from the robot’s center of rotation

5. Odometer is updated with the values obtained in step 4