Syringenator

Generated by Doxygen 1.8.16

Sun Feb 3 2019 06:36:53

1 README

University of Washington

TCES 460

Winter 2019

1.1 Development Team:

Author

Alex Boyle
Ammon Dodson, ammon0@uw.edu
Alex Marlow, alexmarlow117@gmail.com
Jake McKenzie, jake314@uw.edu
Brooke Stevenson

1.2 HypoRobot Assignment

If you've been paying any attention at all to current events you know that a major plague has descended on cities and counties throughout the country in the form of used and discarded hypodermic needles. Countless hours are spent cleaning up this mess. For instance, some schools are forced, for safety reasons, to send staff out to scour the playgrounds prior to children showing up.

Your task this quarter will be to design an autonomous robot that can help automate the arduous and sometimes dangerous job of spotting, retrieving, and disposing of hypodermic syringes.

Your robot will be a prototype, not a fully functional disposal robot, but it will have important technical features necessary on such a robot.

A second point is that we will be dealing with industrial (i.e. dull) syringes. These are typically used to disburse such things glue or solvents. They are commonly used in our labs to glue acrylic parts together. Anyone in the lab with a sharp needle will be immediately disqualified. Even so, if you would rather not design and test with any syringe, you may, with my written permission, use a ballpoint pen, a #2 pencil or a similar object of your choosing.

All testing will be done indoors on a flat surface.

2 Requirements

2.1 Terminology

The following terms are used in this specification:

- The term "autonomous", in this case, means that no commands can be transmitted to your robot from any outside agency (especially from a human or computer or other controller) and all sensors used in the contest must be physically attached to your robot. No wired connections are allowed between any outside agency and your robot.

- The term “course” refers to the area in which the contest takes place.
- The term “tape line” refers to an oval of white tape that runs from a start point around the oval, back to the start point (which is now the finish point). All targets will be placed outside of the oval.
- The term “target” refers to the object you are required to pick up and dispose of (syringe or, alternatively, a pen or pencil).
- The term “decoy” refers an object on the course that is not a target. A decoy will be less than 2 cm tall.
- The term “obstacle” refers to an object on the course that your robot must avoid running into. An obstacle will be at least 15 cm tall. A typical obstacle would be a cardboard box.
- The term “finish the course” will mean that your robot traverses the oval at least once. Note: Your robot will have to leave the tape line to pick up targets, but it should eventually either find another target or return to the tape line. The tape line is your navigation aid.
- The term “contact a target” will mean to touch a target with your pick-up mechanism in such a way as to move it. Note: moving a target with a robot wheel or track does not count as a contact.
- The term “participate” will mean that you either finish the course or contact a target.
- The term “acquire a target” means your robot has reported to its data logger that it has identified a target and reports an accurate position for that target. The term “acquire a decoy” means your robot has reported to its data logger that it has acquired a target that turns out to be a decoy.
- The term “pick up a target” refers to your robot picking up a target off the course surface.
- The term “dispose of a target” refers to your robot placing the target in container on your robot.
- A robot is “stationary” if its wheels are not rotating and its arm is not rotating about its vertical axis.

2.2 Rules of the Game

- You will be given two test runs, one per day over two class periods. The dates will be firmly established by midterm time.
- All tests will be conducted indoors.
- A somewhat different course may be laid out each day. The layout will consist of:
  - A tape line; this will serve as your navigation maker. Since we will be indoors, we won’t have GPS; the tape line will serve as your navigation reference.
  - A number of targets will be placed within 1 meter of the tape line; you will have to leave the tape line to pick up your targets.
  - A number of decoys will be placed within 1 meter of the tape line.
  - A number of obstacles will be placed on the course. If you exactly follow the tape line you will not run into an obstacle; however, you may have to avoid obstacles as you maneuver away from the tape line to pick up targets.
- No human will be allowed on the course during a test run.
- Your robot must be autonomous.
- All test runs will be video ‘taped.’
- The goal is to maximize your score according to the algorithm discussed below. The maximum score you achieve for any one day over the two days will be your final score.

- The scores for the entire class will be rank-ordered.
- You will be allowed ten minutes on the course for each test run. This will be strictly timed.
- Robot
  - You will be provided with
    * A basic robot chassis
    * Two motors with encoders and wheels
    * Two motor controllers (H-bridges)
    * A robotic arm
    * A battery pack with a power distribution unit
    * Distance sensors.
    * Line sensors
    * Data logger with SD card
  - You do not have to use this robot chassis or arm
  - You will need to supply your own processor(s)
  - You will need to supply your own cameras(s) and cables.
  - You may acquire additional mechanical or electronic parts for your robot.
  - If you plan to spend any money on your robot, you must get permission from me in writing first.
  - Your group has a strict budget of $300, including any parts that you have already acquired and use on your robot (e.g., an Arduino).
- Rule 8 applies. Rule 8 comes from the official rules for the annual Race to Alaska (see `https://r2ak.com/official-rules/`). Rule 8 states, and I quote: If we decide it's necessary to consult a lawyer to figure out if you are disqualified or not, you are automatically disqualified. Play by the rules and live up to the spirit of the race. If you get cute and push the boundaries, we'll bring down the hammer.

3 Todo List

File Syringenator.py

TODO: how do we initialize the robot run? a button press? –ABD

Member Syringenator.returnToLine ()

TODO: do we need to check that we actually returned? how do we recover if dead reckoning fails? –ABD

4 Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

5 Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

7 Namespace Documentation

7.1 constants Namespace Reference

Variables

- int ARM_AZIMUTH_MIN = 0
- int ARM_AZIMUTH_MAX = 0
- int ARM_RANGE_MIN = 0
- int ARM_RANGE_MAX = 0
- int ARM_ORIENT_MIN = 0
- int ARM_ORIENT_MAX = 0
- int PICKUP_X_MIN = 0
- int PICKUP_X_MAX = 0
- int PICKUP_Y_MIN = 0
- int PICKUP_Y_MAX = 0
- int ARDUINO_NULL = 0x00
- int ARDUINO_STATUS_ACK = 0x01
- int ARDUINO_STATUS_READY = 0x02
- int ARDUINO_STATUS_PICK_FAIL = 0x03
- int ARDUINO_STATUS_PICK_SUCCESS = 0x04
- int ARDUINO_STATUS_ARM_FAULT = 0x05
- int ARDUINO_STATUS_OBSTACLE = 0x06
- int ARDUINO_ROTATE = 0x10
- int ARDUINO_MOVE = 0x11
- int ARDUINO_LINE_FOLLOW = 0x12
- int ARDUINO_ARM_PARK = 0x20
- int ARDUINO_ARM_DISPOSE = 0x21
- int ARDUINO_ARM_PICKUP = 0x22

7.1.1 Variable Documentation

7.1.1.1 ARDUINO_ARM_DISPOSE

```
int constants.ARDUINO_ARM_DISPOSE = 0x21
```

7.1.1.2 ARDUINO_ARM_PARK

```
int constants.ARDUINO_ARM_PARK = 0x20
```

7.1.1.3 ARDUINO_ARM_PICKUP

```
int constants.ARDUINO_ARM_PICKUP = 0x22
```

7.1.1.4 ARDUINO_LINE_FOLLOW

```
int constants.ARDUINO_LINE_FOLLOW = 0x12
```

7.1.1.5 ARDUINO_MOVE

```
int constants.ARDUINO_MOVE = 0x11
```

7.1.1.6 ARDUINO_NULL

```
int constants.ARDUINO_NULL = 0x00
```

7.1.1.7 ARDUINO_ROTATE

```
int constants.ARDUINO_ROTATE = 0x10
```

7.1.1.8 ARDUINO_STATUS_ACK

```
int constants.ARDUINO_STATUS_ACK = 0x01
```

7.1.1.9 ARDUINO_STATUS_ARM_FAULT

```
int constants.ARDUINO_STATUS_ARM_FAULT = 0x05
```

7.1.1.10 ARDUINO_STATUS_OBSTACLE

```
int constants.ARDUINO_STATUS_OBSTACLE = 0x06
```

7.1.1.11 ARDUINO_STATUS_PICK_FAIL

```
int constants.ARDUINO_STATUS_PICK_FAIL = 0x03
```

7.1.1.12 ARDUINO_STATUS_PICK_SUCCESS

```
int constants.ARDUINO_STATUS_PICK_SUCCESS = 0x04
```

7.1.1.13 ARDUINO_STATUS_READY

```
int constants.ARDUINO_STATUS_READY = 0x02
```

7.1.1.14 ARM_AZIMUTH_MAX

```
int constants.ARM_AZIMUTH_MAX = 0
```

7.1.1.15 ARM_AZIMUTH_MIN

```
int constants.ARM_AZIMUTH_MIN = 0
```

7.1.1.16 ARM_ORIENT_MAX

```
int constants.ARM_ORIENT_MAX = 0
```

7.1.1.17 ARM_ORIENT_MIN

```
int constants.ARM_ORIENT_MIN = 0
```

7.1.1.18 ARM_RANGE_MAX

```
int constants.ARM_RANGE_MAX = 0
```

7.1.1.19 ARM_RANGE_MIN

```
int constants.ARM_RANGE_MIN = 0
```

7.1.1.20 PICKUP_X_MAX

```
int constants.PICKUP_X_MAX = 0
```

7.1.1.21 PICKUP_X_MIN

```
int constants.PICKUP_X_MIN = 0
```

7.1.1.22 PICKUP_Y_MAX

```
int constants.PICKUP_Y_MAX = 0
```

7.1.1.23 PICKUP_Y_MIN

```
int constants.PICKUP_Y_MIN = 0
```

7.2 Syringenator Namespace Reference

The top-level Pi program.

Classes

- class Target

Functions

- def arduinoSend (bytes)

    Send serial data to the arduino.
- def arduinoReceive ()

    Wait some fixed time for the arduino to send one or more bytes.
- def scan ()

    A routine to take a picture and report back the closest target The Computer vision routine must be able to handle multiple targets in the image.
- def moveCloser (t)

    Move the robot closer to the given target.
- def pickUp (t)

    Attempt to pickup and dispose the target.
- def returnToLine ()

    signl the arduino to return to the line.
- def lineFollow ()

    Follow the line.
- def canBePicked (t)

    A routine to determine if the target is in position to be picked up.

Variables

- bool onTheLine = True

    boolean indicating whether we are on the line
- def target = scan()

7.2.1 Function Documentation

7.2.1.1 arduinoReceive()

```
def Syringenator.arduinoReceive ( )
```

Returns

an list of bytes

7.2.1.2 arduinoSend()

```
def Syringenator.arduinoSend (
            bytes )
```

Parameters

bytes	one or more bytes of data to send to the arduino

7.2.1.3 canBePicked()

```
def Syringenator.canBePicked (
             t )
```

Calculates whether the center of the target bounding box is in the pickup area.

Returns

a boolean

7.2.1.4 lineFollow()

```
def Syringenator.lineFollow ( )
```

this routine simply signals the arduino to execute its lineFollow() routine

Returns

None

7.2.1.5 moveCloser()

```
def Syringenator.moveCloser (
             t )
```

The moveCloser() routine attempts to aproach the target by relatively small increments. Because the move routines may be interrupted by the obstacle avoidance ISRs and the risk of jambing the wheels etc. we cannot expect to be able to approch successfully on the first try. Hence moveCloser() should only move a relatively short distance before exiting to allow another loop through the scan cycle.

Should we spend effort trying to avoid running over decoys here?

This routine should check for ARDUINO_STATUS_OBSTACLE. then what?

This routine is likely where we will have the most issues. –ABD

Parameters

t	a Target object containing the location of the target to be approched

Returns

None

7.2.1.6 pickUp()

```
def Syringenator.pickUp (
        t )
```

This routine must determine orientation of the target. If this is not done by some OpenCV magic we can attempt it here using the raw image data and the bounding box.

Divide the longer dimension of the bounding box by some constant divisor. Scan along each of those raster lines twice. On the first pass calculate an average brightness (RGB values can be summed). The second pass will pick out points of greatest brightness. Find the centers of clustered bright pixeles. We now have a set of points in cartesian space. Have Jake find the slope of the line of best fit.

The center can be estimated as the center of the bounding box, or the center of the points, the mean of both, etc.

Once the values for x, y, and m have been determined they will have to pass through a calibration transform to determine the arm a, r, o values. –ABD

Parameters

t	a Target object containing the raw bitmap data

Returns

None

7.2.1.7 returnToLine()

```
def Syringenator.returnToLine ( )
```

Todo TODO: do we need to check that we actually returned? how do we recover if dead reckoning fails? –ABD

Returns

None

7.2.1.8 scan()

```
def Syringenator.scan ( )
```

It would be best if all targets are reported. Then this routine will determine the closest one to pursue. –ABD

Returns

a target object

7.2.2 Variable Documentation

7.2.2.1 onTheLine

```
bool Syringenator.onTheLine = True
```

7.2.2.2 target

```
def Syringenator.target = scan()
```

8 Class Documentation

8.1 Syringenator.Target Class Reference

The documentation for this class was generated from the following file:

- src/Syringenator.py

9 File Documentation

9.1 README.md File Reference

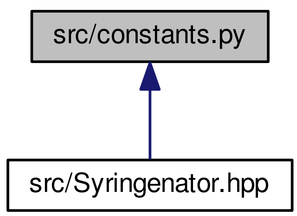
9.2 requirements.md File Reference

9.3 src/constants.dox File Reference

9.4 src/constants.py File Reference

Includes constants used by both the arduino sketch and the the python script.

This graph shows which files directly or indirectly include this file:



Namespaces

- constants

Variables

- int constants.ARM_AZIMUTH_MIN = 0
- int constants.ARM_AZIMUTH_MAX = 0
- int constants.ARM_RANGE_MIN = 0
- int constants.ARM_RANGE_MAX = 0
- int constants.ARM_ORIENT_MIN = 0
- int constants.ARM_ORIENT_MAX = 0
- int constants.PICKUP_X_MIN = 0
- int constants.PICKUP_X_MAX = 0
- int constants.PICKUP_Y_MIN = 0
- int constants.PICKUP_Y_MAX = 0
- int constants.ARDUINO_NULL = 0x00
- int constants.ARDUINO_STATUS_ACK = 0x01

- int constants.ARDUINO_STATUS_READY = 0x02
- int constants.ARDUINO_STATUS_PICK_FAIL = 0x03
- int constants.ARDUINO_STATUS_PICK_SUCCESS = 0x04
- int constants.ARDUINO_STATUS_ARM_FAULT = 0x05
- int constants.ARDUINO_STATUS_OBSTACLE = 0x06
- int constants.ARDUINO_ROTATE = 0x10
- int constants.ARDUINO_MOVE = 0x11
- int constants.ARDUINO_LINE_FOLLOW = 0x12
- int constants.ARDUINO_ARM_PARK = 0x20
- int constants.ARDUINO_ARM_DISPOSE = 0x21
- int constants.ARDUINO_ARM_PICKUP = 0x22

9.4.1 Detailed Description

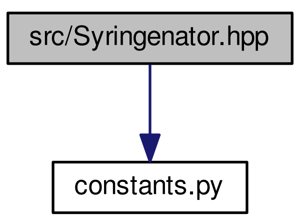
The formatting used in this file is a quick and dirty hack. There are better ways to do this.

9.5 src/Syringenator.hpp File Reference

Arduino controller code.

```
#include "constants.py"
```

Include dependency graph for Syringenator.hpp:



Functions

- void lineDetector_ISR (void)

  A function to respond to a line detector being triggered.
- void obstacleDetector_ISR (void)

  A function to respond to a detected obstacle while under locamotion.
- void motorEncoder_ISR (void)

  Motor encoder ISR.
- void serialCommunication_ISR (void)

A function to handle incomming communication from the pi.
- void moveRotate (int ticks)

  Rotate the robot around central axis rotate by running both motors at the same speed in opposite directions.
- void moveStraight (int ticks)

  Move the robot forward or reverse.
- void moveLineFollow (void)

  Routine to follow the guide-line for some fixed interval.
- void armPark (void)

  Move the arm to its parking position.
- void armDispose (void)

  Routine to dispose of a syringe once it has been picked.
- bool armPick (byte azimuth, byte range, byte orientation)

  Routine to attempt target pickup.

9.5.1 Detailed Description

–ABD

9.5.2 Function Documentation

9.5.2.1 armDispose()

```
void armDispose (
            void  )
```

9.5.2.2 armPark()

```
void armPark (
            void  )
```

The parking position needs to leave a clear view of the pickup area, but also should move the center of gravity as far forward as possible to reduce drive wheel slippage.

9.5.2.3 armPick()

```
bool armPick (
            byte azimuth,
            byte range,
            byte orientation )
```

This routine should attempt to close the claw completely and detect if an object as actually been grabbed. parameters should be bytes because they will have to be transmitted over serial from the pi. Ranges on these values TBD as convenient for the arm software, but must be recorded in the system constants file. –ABD

Parameters

azimuth	arm azimuth value
range	distance to the target
orientation	rotation of the target

Returns

true on successful pick, false otherwise.

9.5.2.4 lineDetector_ISR()

```
void lineDetector_ISR (
            void  )
```

The line detectors are mounted forward and inboard of the wheels. This function needs to reorient the robot to clear the sensor, but also to prevent the line from being hit again.

The simplest way to do this is to rotate the opposite wheel forward until the sensor clears. Because the sensor is forward of the wheel it will rotate away from the line as the opposite wheel moves forward. This should work as long as the curvature of the line is not too great.

This may need to be two routines, one for each sensor –ABD

9.5.2.5 motorEncoder_ISR()

```
void motorEncoder_ISR (
            void  )
```

9.5.2.6 moveLineFollow()

```
void moveLineFollow (
            void  )
```

This function assumes that we are already over the line

9.5.2.7 moveRotate()

```
void moveRotate (
            int ticks )
```

Parameters

ticks	sign indicates direction of rotation: positive is rotation to the right. magnitude indicates the number of encoder ticks on each motor.

9.5.2.8 moveStraight()

```
void moveStraight (
            int ticks )
```

Parameters

ticks	number of encoder ticks to move. Sign indicates direction: positive is forward.

9.5.2.9 obstacleDetector_ISR()

```
void obstacleDetector_ISR (
            void )
```

There may be two cases to handle: whether we are line following, or aproaching. If we are line following we need to ensure that we don't lose the line while avoiding the obstacle.

This may need to be multiple routines, one for each sensor –ABD

9.5.2.10 serialCommunication_ISR()

```
void serialCommunication_ISR (
            void )
```

9.6 src/Syringenator.py File Reference

This is the main control script.

Classes

- class Syringenator.Target

Namespaces

- Syringenator

    The top-level Pi program.

Functions

- def Syringenator.arduinoSend (bytes)

  Send serial data to the arduino.
- def Syringenator.arduinoReceive ()

  Wait some fixed time for the arduino to send one or more bytes.
- def Syringenator.scan ()

  A routine to take a picture and report back the closest target The Computer vision routine must be able to handle multiple targets in the image.
- def Syringenator.moveCloser (t)

  Move the robot closer to the given target.
- def Syringenator.pickUp (t)

  Attempt to pickup and dispose the target.
- def Syringenator.returnToLine ()

  signl the arduino to return to the line.
- def Syringenator.lineFollow ()

  Follow the line.
- def Syringenator.canBePicked (t)

  A routine to determine if the target is in position to be picked up.

Variables

- bool Syringenator.onTheLine = True

  boolean indicating whether we are on the line
- def Syringenator.target = scan()

9.6.1 Detailed Description

It will run on the Raspberry Pi and direct all robot operations.

Todo TODO: how do we initialize the robot run? a button press? –ABD

Index