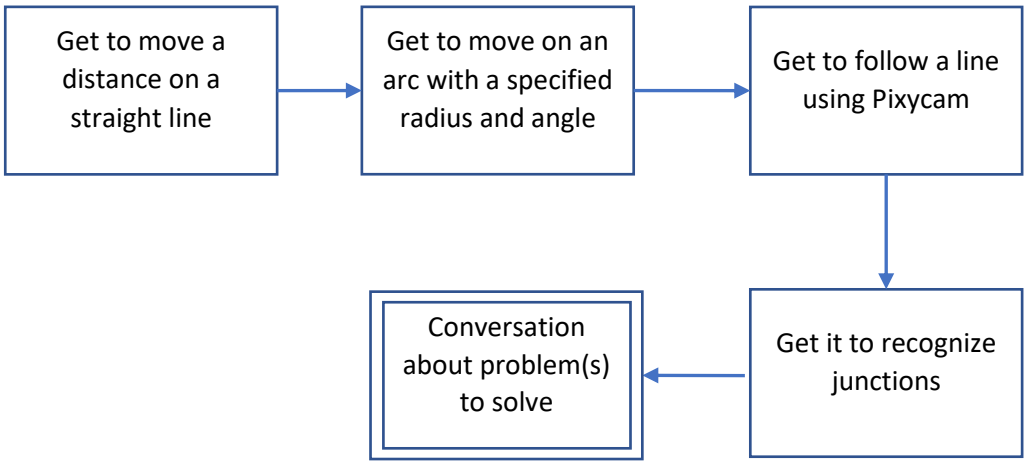
Harry Investigation Guidance

Here's a suggested workflow



Some House Rules

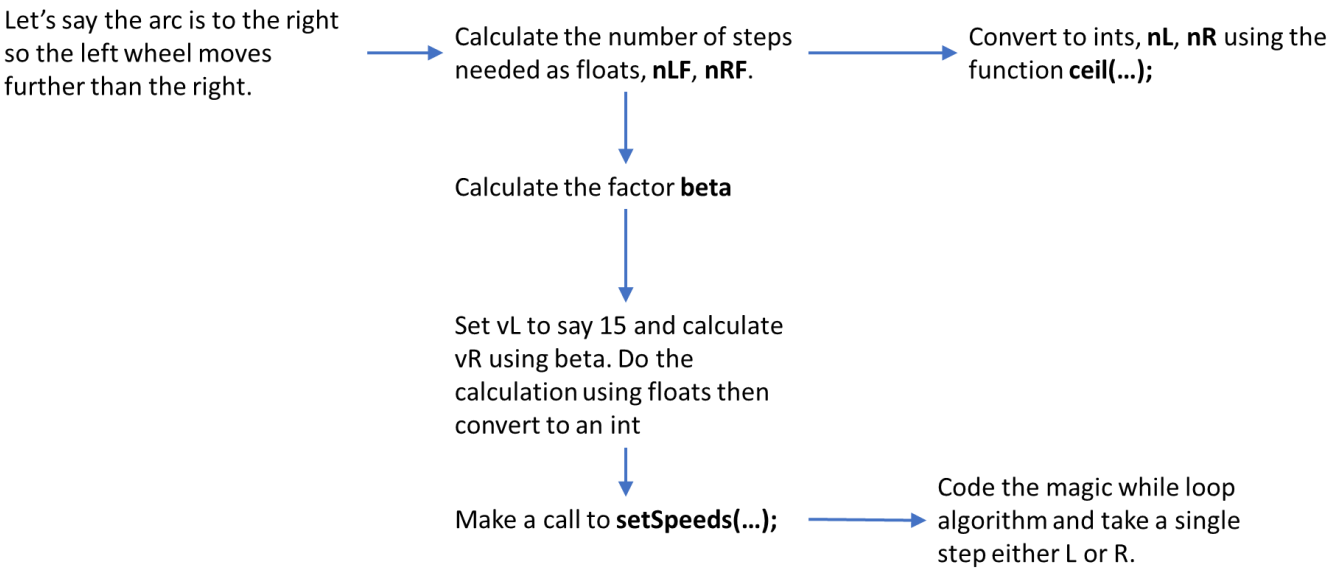
- When using floats (in declarations and assignments) use the decimal point 100.0
- Declare Pixycam variables following their API
- Template sketches
  - Have a main file 3402_R_
  - Call the library CBPFBO_StepperA.h or CBPFBO_StepperAX.h in libraries folder
  - Use a second tab 3402_Helper1 which contains **cprintf(….);**
- Library files are in **portable > sketchbook > libraries**
- Use the **portable** Arduino folder

Moving along a straight line

- Sketch 3402_R_Straight_Line.ino
  - variable **dx** (library) is the distance a wheel moves in mm for one step
  - function **setSpeeds(…)** is found in **CBPFBO_StepperAX.h**
  - function **stepMotors(…)** is also found there
  - use a for or while loop to take steps, each iteration take 1 step for left & right
- Complete the sketch to get the robot to move forward a given distance in mm.

Moving along an arc

- Sketch 3402_R_Arc
  - You should specify a desired distance in mm and an angle in degrees, then convert to rads.
  - The 'algorithm' which you need to code is outlined below and details are provided at this link https://colin-price.wbs.uni.worc.ac.uk/Courses_2021_22/Comp2403/CBP_Notes_Book/Ch1_Kinematics_StepperMotors.pdf
- Complete and test the sketch
  - Some lines need uncommenting
  - Other lines need adding, see below and link above.

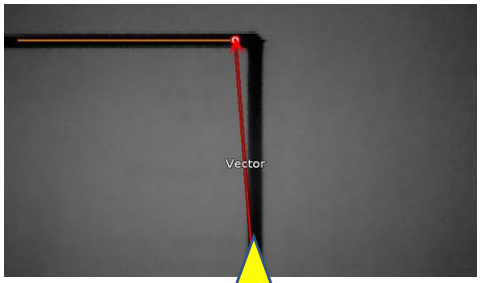


Line Following

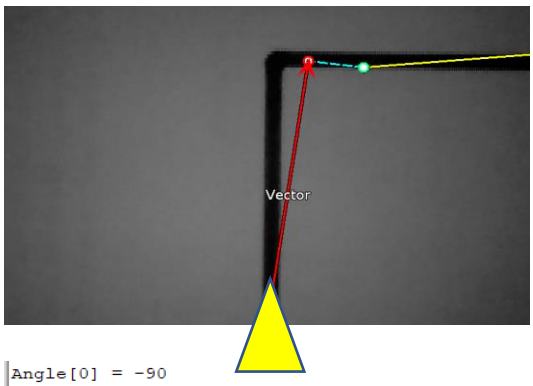
- Sketch 3402_R_Line_Follow
  - Write a function **getErrorX()**. You need to use the value of **x0** from the vector tail and the width of the camera when it is line mode (I seem to recall this is 40 pels). Don't forget to normalize the result
  - Use the function **moveABit(float error, float deltaT);** in the library CBP_FBO_StepperA. Play around with deltaT, sometimes I set this to 0.5, other times to 0.8.
- Complete and test out the sketch

Detecting Intersections

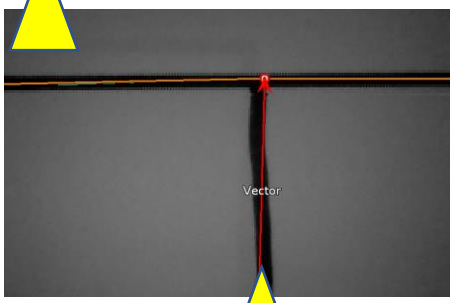
- Sketch 3402_R_Junction_Test will let you explore Pixy's intersection API Here's what I got for a left turn, a right turn and a tee. Didn't do a crossing.



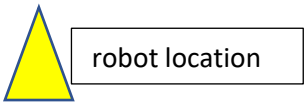

```
Angle[0] = -176
Angle[1] = 90
Angle[2] = 0
Angle[3] = 0
```




```
Angle[0] = -90
Angle[1] = 177
Angle[2] = 0
Angle[3] = 0
```



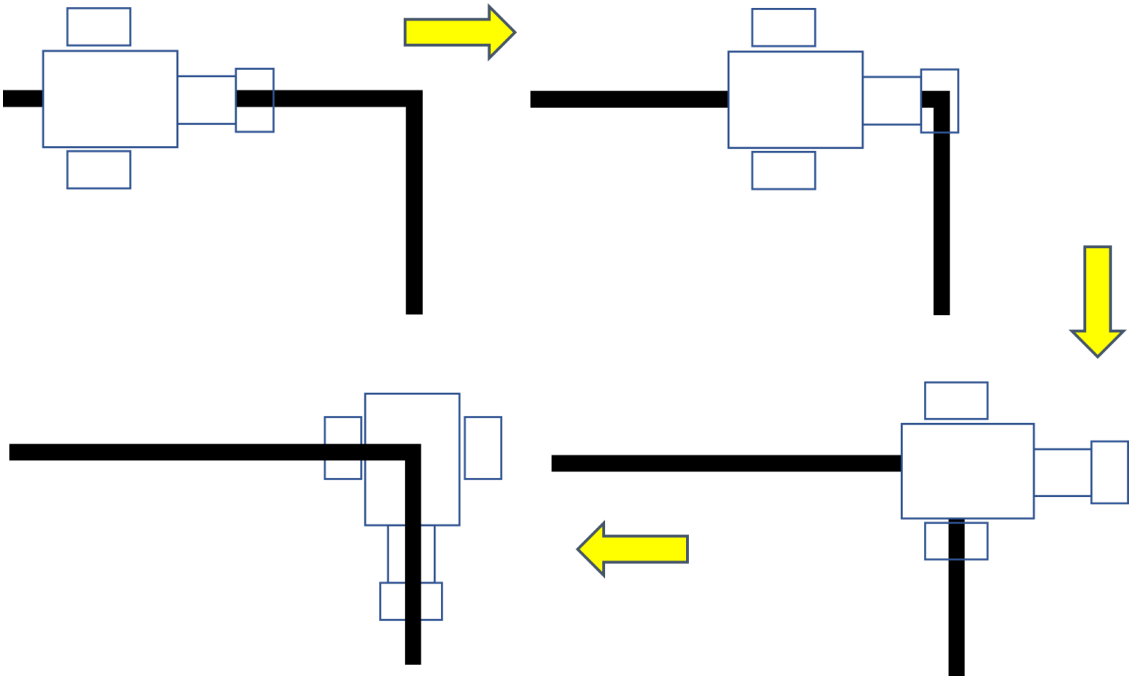

```
Angle[0] = -90
Angle[1] = 90
Angle[2] = 180
Angle[3] = 0
```



- Look at the *range* of angles the function returns, not exactly 90 or -90.

- Now open up the sketch **3402_R_Junction_React**
  - Look over the function **junctionSearch(…)** which identifies the junction (LEFT, RIGHT, TEE, NONE) which are defined in CBPFBO_StepperA.h
  - This calls **detectCorners(…)** which is found in CBPFBO_StepperA.cpp. You can see how we have dealt with the uncertain angles returned by the PixyCam API and output clean exact angle

Now time to code the robot to detect junctions and to turn. Let's have a look at one approach. Here's what the robot should do. It starts off top left following the line. Then top right PixyCam detects the junction. So the robot must move forwards until the centre of its wheel is exactly above the junction. Then it spins 90 degrees about the centre of its axle. At this point it has navigated the junction and moves on to the next challenge.



The following functions are available in the library. Some, you've already seen.

void straightLine(int dist, int vMax, bool upRamp, bool downRamp);	moves a distance with a max vely and can ramp its speed up at the start and down at the end. Both can be turned on or off.
int detectCorners(int16_t angs[]);	returns junction type when fed with an array of PixyCam intersection angles
void moveABit(float error, float deltaT);	follows a line when provided with the perceived error and some time constant

- Now code robot behaviour to follow straight lines and navigate intersections
  - Think about what code architecture to use.