
DTU

34755 Building dependable robots

- Today
  - Course status
    - Questions before we start
    - Group forming – end today
  - Software intro
    - Line sensor
    - Follow line
  - Group work
  - Solve the quiz of the day

J. Christian Andersen
PCAS
DTU Electro

34755 – Building dependable robot systems

2025

Project plan hand-in

Half way Challenge

DTU Robocup competition

Competition

Project work

Report

time

1 2 3 4 5 6 7 Easter 8 9 10 11 12 13

Report

Intro, group, investigate

More info. Project Analysis

Project planning Groupwork

Test and debug (1h lecture)

Reliability (1h lecture)

Course eval

Skylab lecture

?

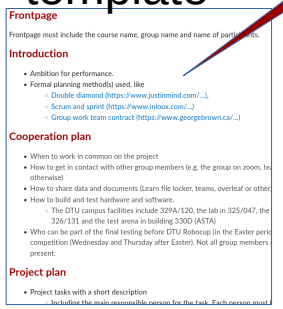
Course eval

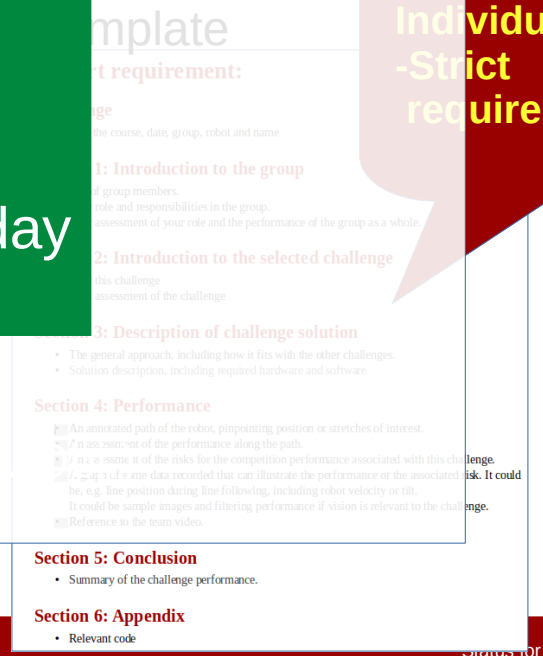
Individual report -Strict requirements

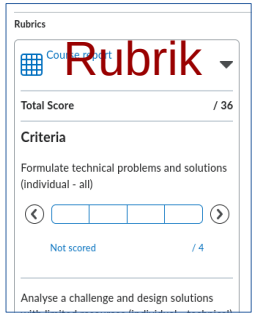
Any outstanding questions

- Quizzes?
- Groupwork?
- See also FAQ
- Group forming end today
- Anything else?

template



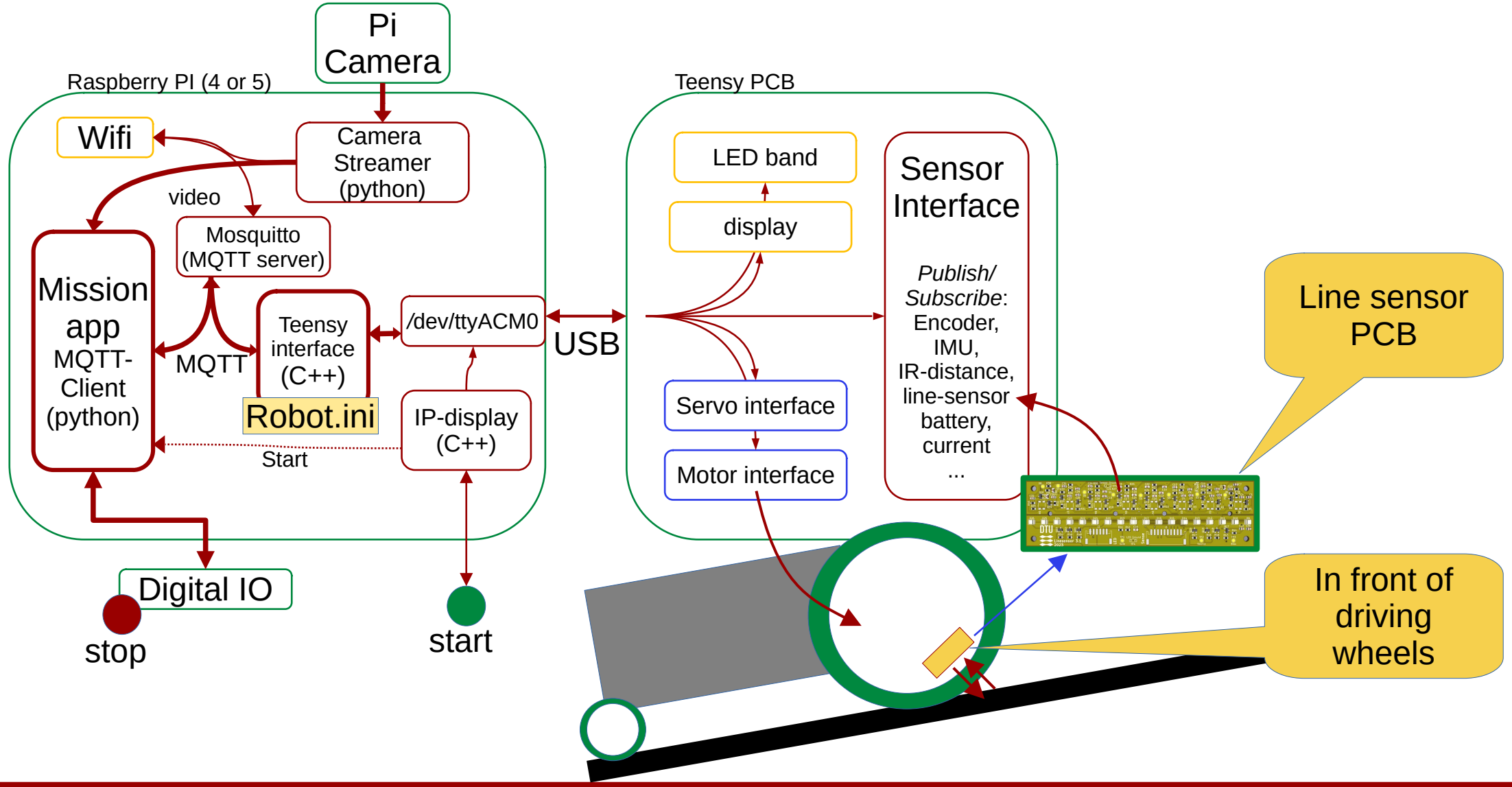




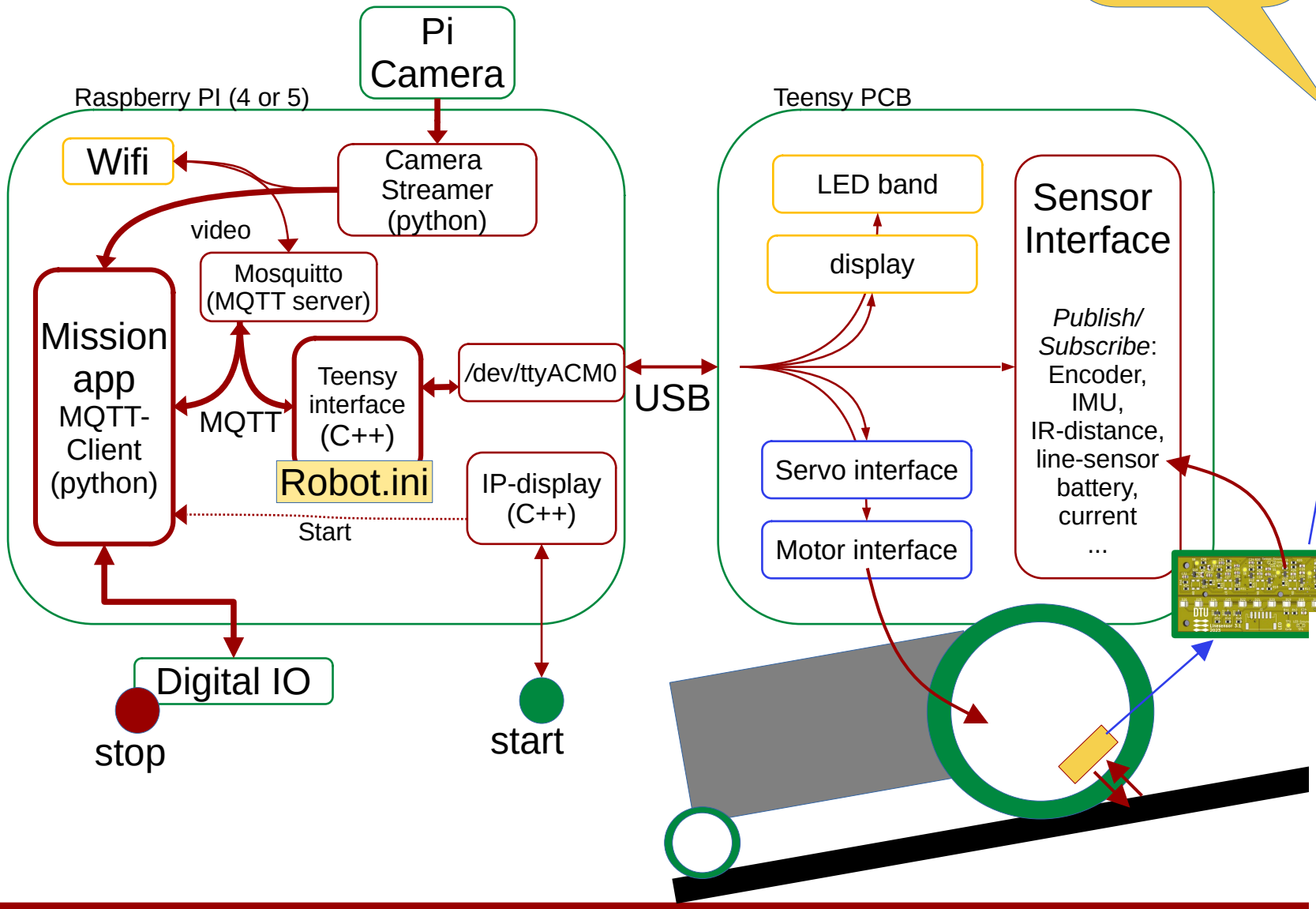
34755 Building dependable robots

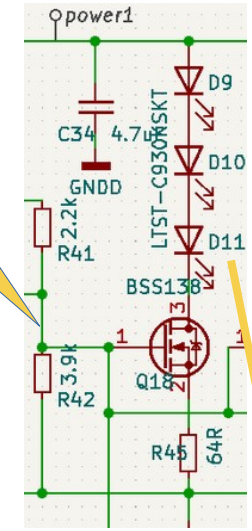
- Today
  - Course status
    - Questions before we start
    - Group forming – end today
  - Software intro
    - Line sensor
    - Follow line
  - Group work
  - Solve the quiz of the day

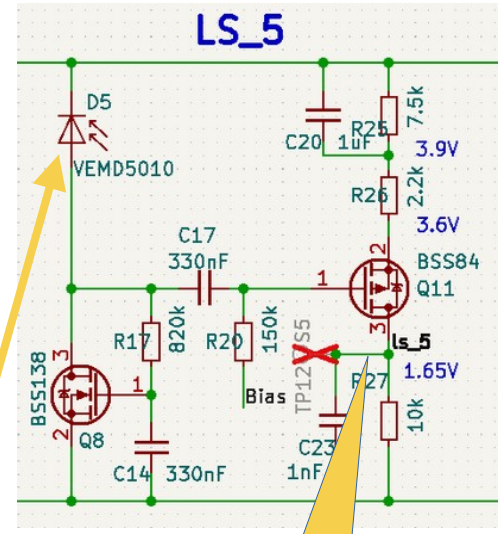
Software – line sensor

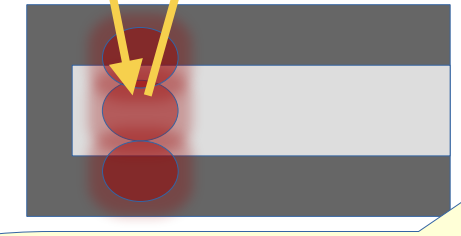


Software – line sensor





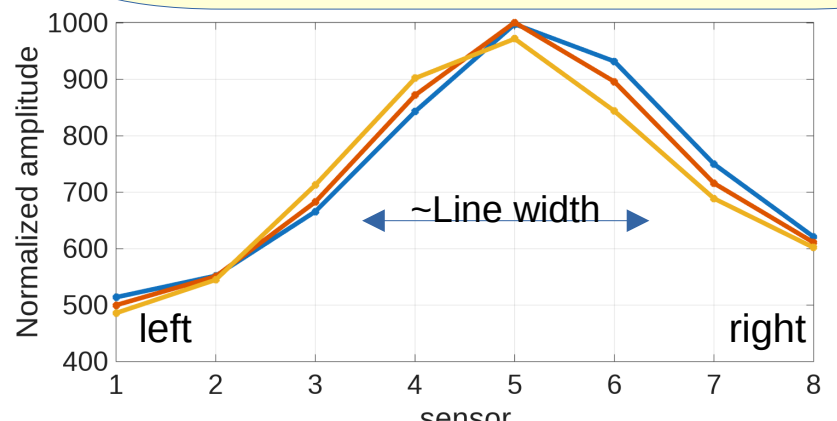




Pulse
0.5ms on
0.5ms off

Square value To A/D

Square amplitude, normalized (0..1000)
1770896214.815 514 552 666 843 998 932 750 621
1770896214.825 500 551 683 872 1001 896 716 611
1770896214.836 486 545 713 902 972 844 689 602

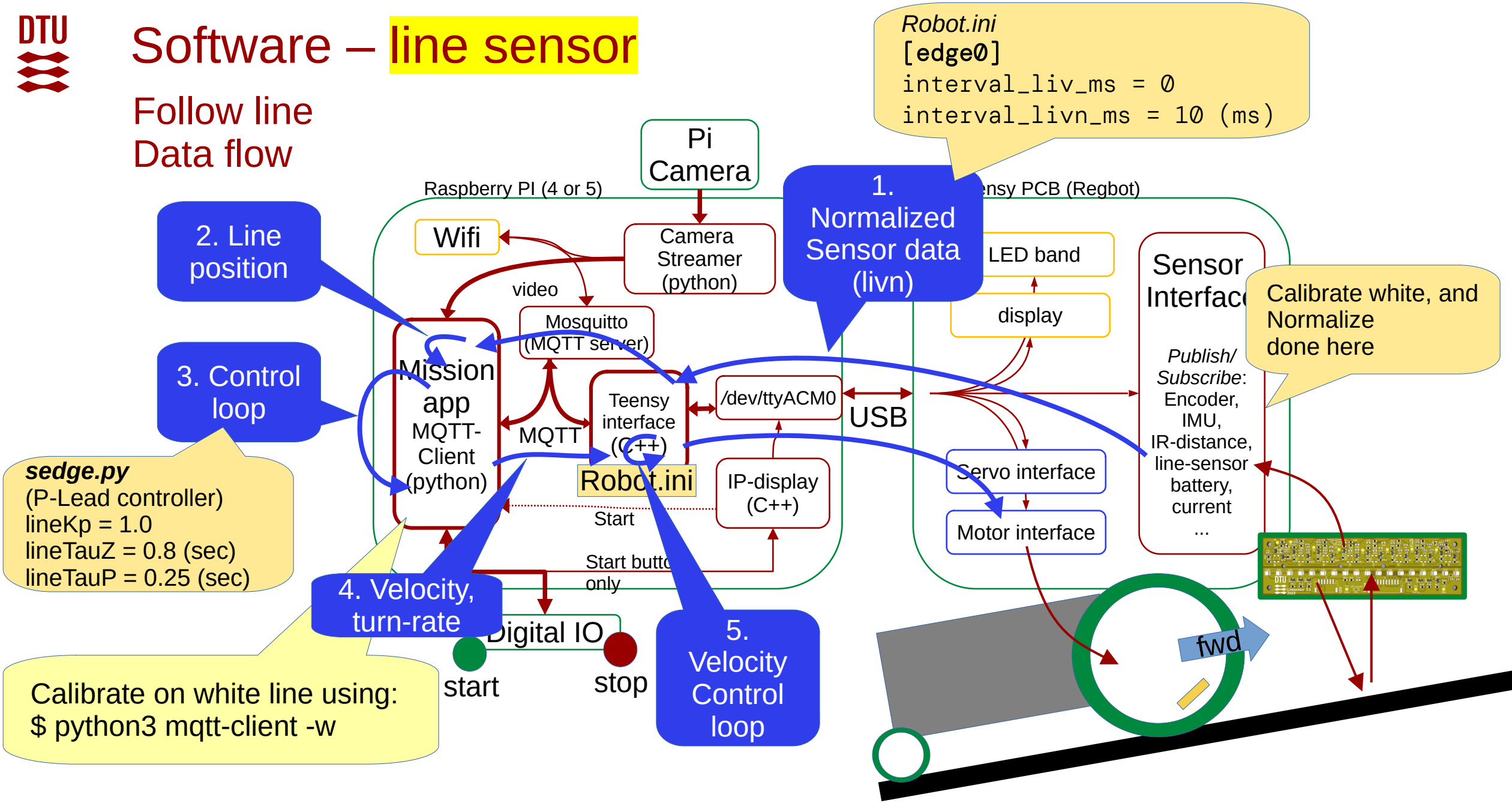


34755 Building dependable robots

- Today
  - Course status
    - Questions before we start
    - Group forming – end today
  - Software intro
    - Line sensor
    - Follow line
  - Group work
  - Solve the quiz of the day

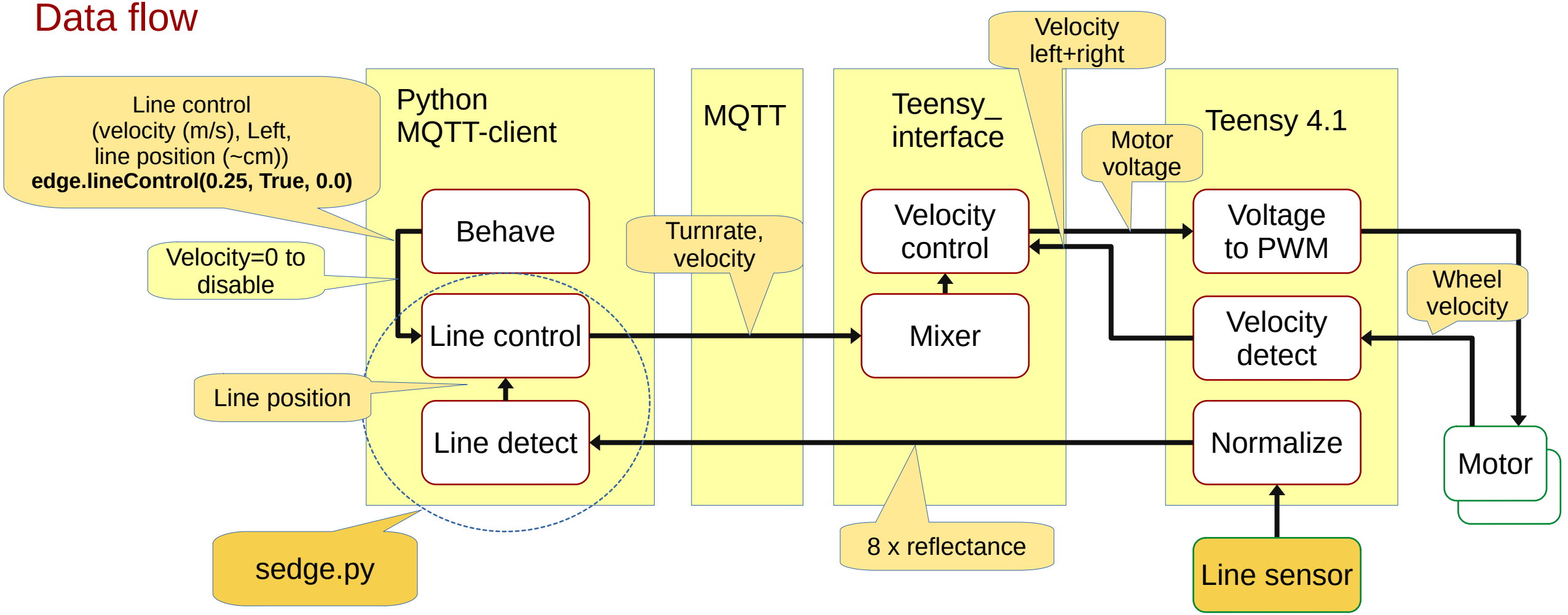
Software – line sensor

Follow line
Data flow



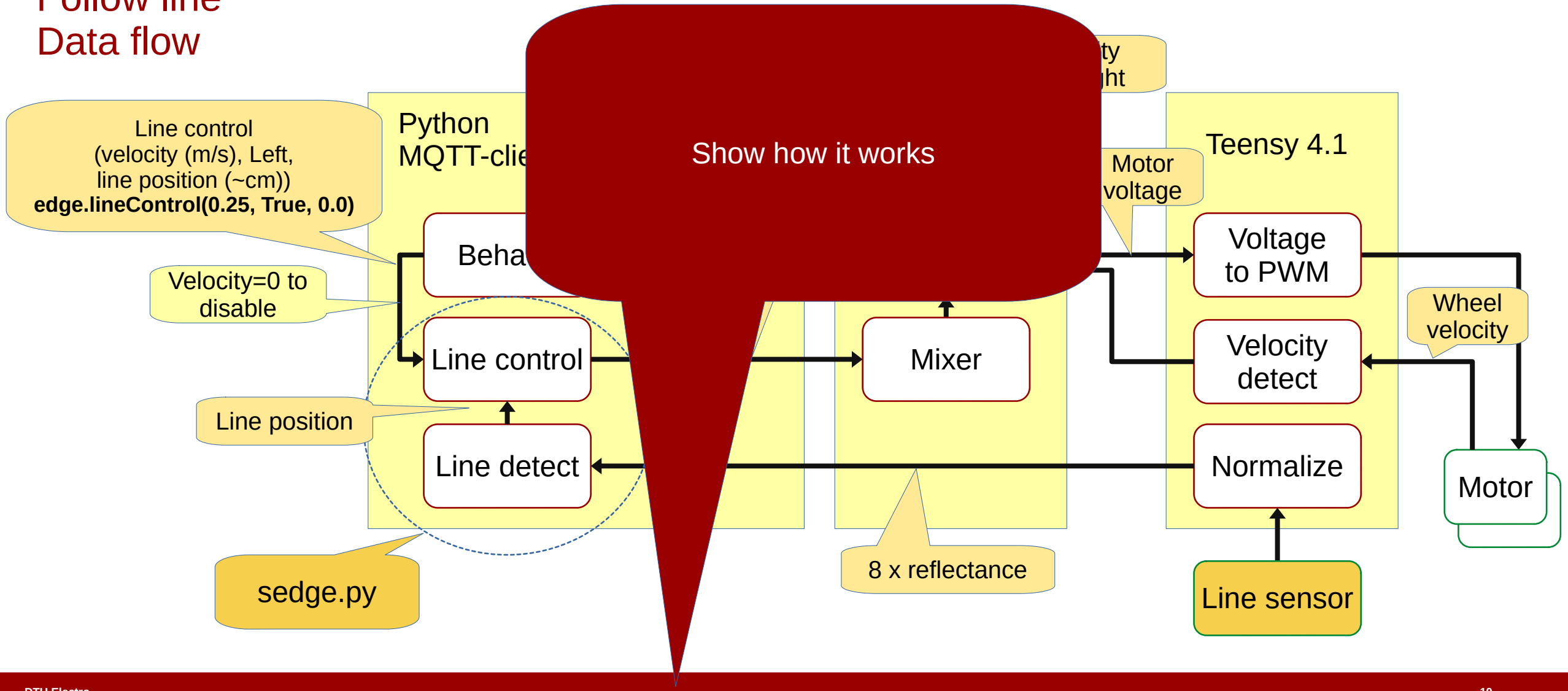
Software – line sensor

Follow line
Data flow



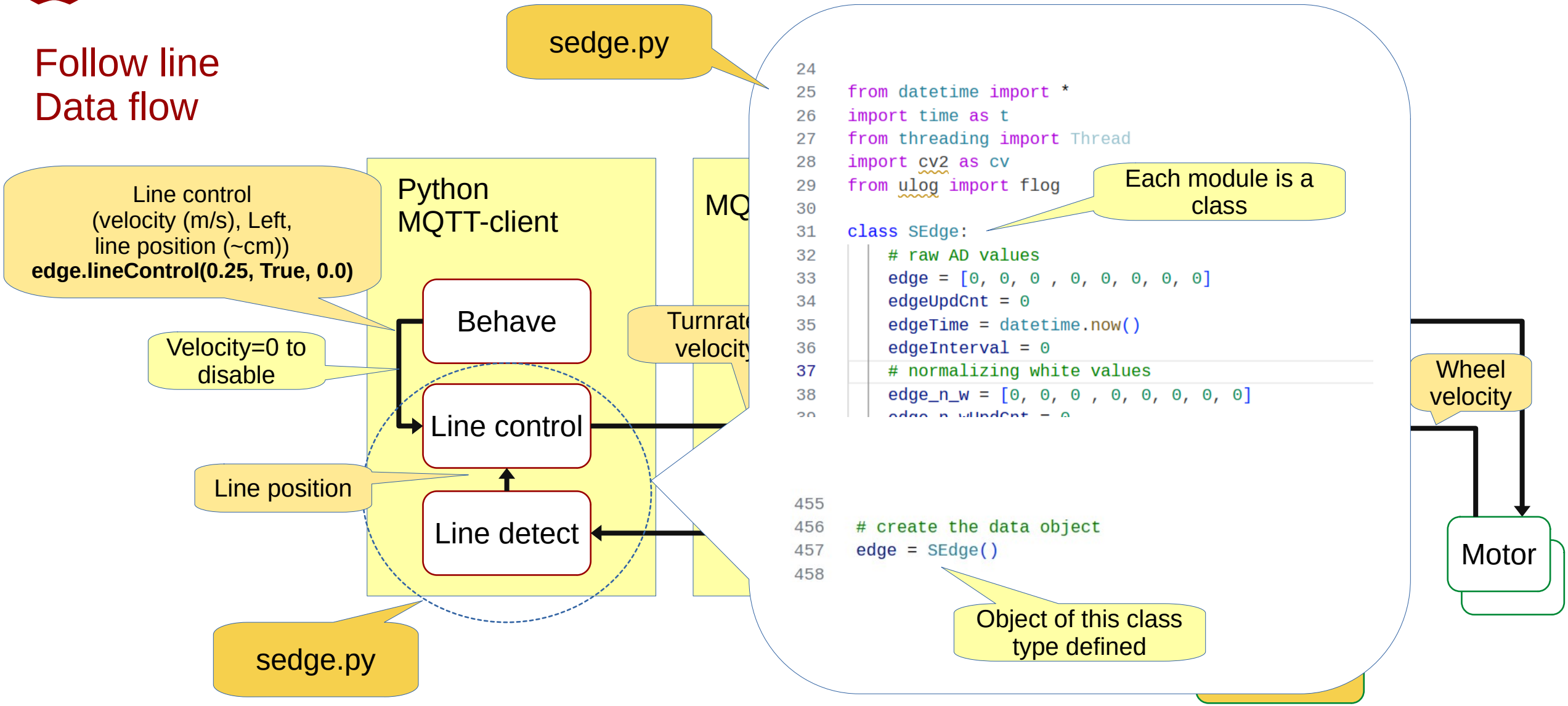
Software – line sensor

Follow line
Data flow



Software – line sensor

Follow line
Data flow

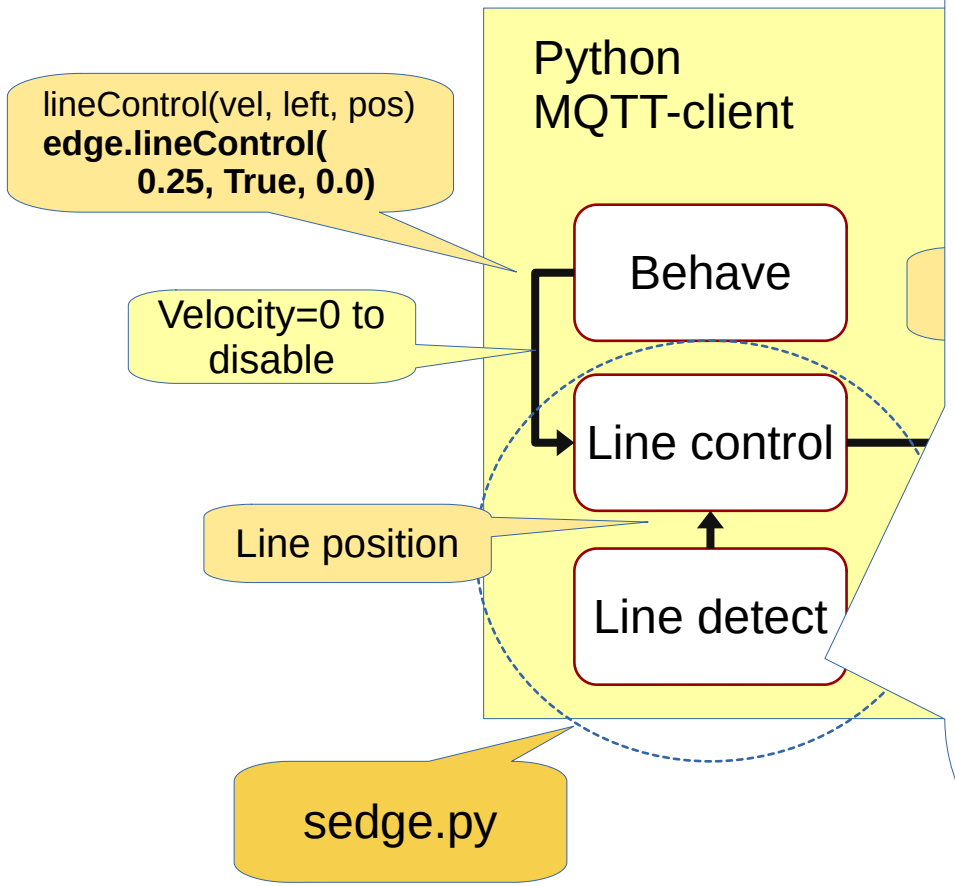



```
24
25  from datetime import *
26  import time as t
27  from threading import Thread
28  import cv2 as cv
29  from ulog import flog
30
31  class SEdge:
32      # raw AD values
33      edge = [0, 0, 0 , 0, 0, 0, 0, 0]
34      edgeUpdCnt = 0
35      edgeTime = datetime.now()
36      edgeInterval = 0
37      # normalizing white values
38      edge_n_w = [0, 0, 0 , 0, 0, 0, 0, 0]
```

```
455
456  # create the data object
457  edge = SEdge()
458
```

Software – line sensor

MQTT message:
Topic = ”robobot/drive/**T0/livn**”
Msg = ”1770896214.825 500 551 683 872 1001 896 716 611”
Sent to edge.decode(…).



sedge.py

```
31  class SEdge:
        def decode(self, topic, msg):
198         # decode MQTT message
199         used = True
200 >       if topic == "T0/liv": # raw AD value...
221         elif topic == "T0/livn": # normalized after calibration rang
222           from uservice import service
223           gg = msg.split(" ")
224           if (len(gg) >= 4):
225             t0 = self.edge_nTime;
226             self.edge_nTime = datetime.fromtimestamp(float(gg[0]))
227             self.edge_n[0] = int(gg[1])
228             self.edge_n[1] = int(gg[2])
229             self.edge_n[2] = int(gg[3])
230             self.edge_n[3] = int(gg[4])
231             self.edge_n[4] = int(gg[5])
232             self.edge_n[5] = int(gg[6])
233             self.edge_n[6] = int(gg[7])
234             self.edge_n[7] = int(gg[8])
235             t1 = self.edge_nTime;

244             #
245             # calculate line values based on new values
246             self.LineDetect()
247             #
248             # use to control, if active
249             if self.lineCtrl:
250               self.followLine()
```

Decode from string to integer

Find edges

Follow edge (if active)

Software – line sensor

Follow line
Data flow

sedge.py

Python
MQTT-client

lineControl(vel, left, pos)
edge.lineControl(
0.25, True, 0.0)

Behave

Velocity=0 to disable

Line control

Line position

Line detect

sedge.py

Called on update (10ms)

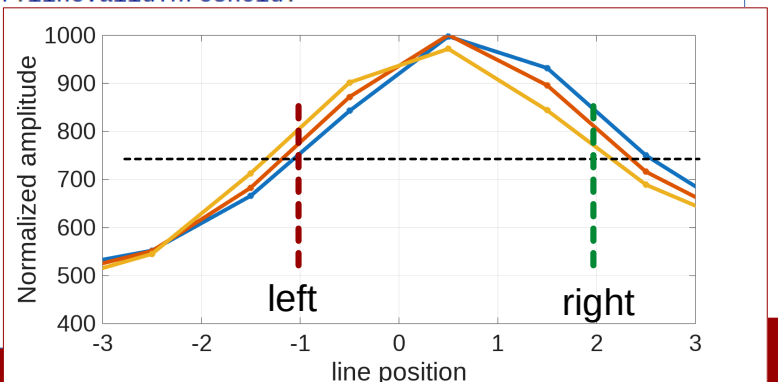
```
276     def LineDetect(self):
277       sum = 0
278       posSum = 0
279       high = int(1)
280       # find levels (and average)
281       # using normalised readings (0 (no reflection) to 1000 (calibrated white)))
282       for i in range(8):
283         sum += self.edge_n[i] # for average
284         if self.edge_n[i] > high:
285           high = self.edge_n[i] # most bright value (floor level)
286       self.high = high # most white level
290       # detect if we have a crossing line
291       self.crossingLine = self.average >= self.crossingThreshold
292       # is line valid (high above threshold)
293       self.lineValid = self.high >= self.lineValidThreshold
296       if self.lineValid:
297         posLeft = -3.5 # max left
298         if self.edge_n[0] < self.lineValidThreshold:
299           posLeft = -3 # between sensor 1 an
300           for i in range(1,8):
301             if self.edge_n[i] < self.lineValidThreshold:
302               posLeft += 1;
303             else:
304               break;
305         posRight = 3.5 # max right
306         if self.edge_n[7] < self.lineValidThreshold:
307           posRight = 3 # may be between sensor 8 and 7 or more left
308           for i in range(1,8):
309             if self.edge_n[7-i] < self.lineValidThreshold:
310               posRight -= 1;
311             else:
312               break;
313         self.posLeft = posLeft
314         self.posRight = posRight
315       else:
316         # just keep old value
```

Most bright

Line valid
> 750

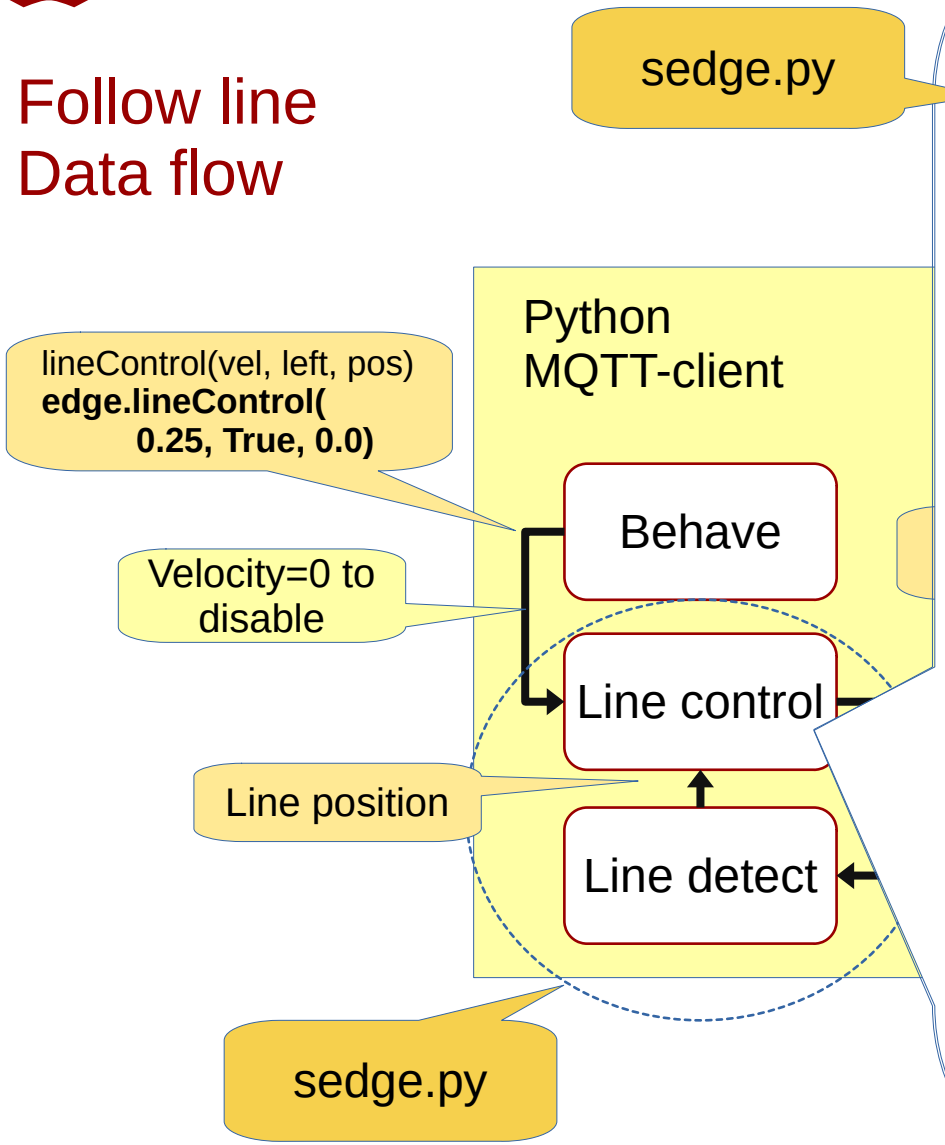
Iterate from left to right

Increase pos as
long as ”no line”



Software – line sensor

Follow line
Data flow




```
347  def followLine(self):
348      from uservice import service
349      # some parameters depend on sample time, adjust
350      # print(f"LineCtrl:: sample time {self.edge_nInterval}")
351      if abs(self.edge_nInterval - self.edgeIntervalSetup) > 2.0: # ms
352          self.PIDrecalculate()
353          self.edgeIntervalSetup = self.edge_nInterval
354      if self.followLeft:
355          e = self.refPosition - self.posLeft
356      else:
357          e = self.refPosition - self.posRight
358      # when line (posLeft or posRight) is to (much) to the right edge position is positive.
359      # The robot is thus too much to the left.
360      # To correct we need a negative turn rate (CV),
361      # so sign of e is OK
362      #
363      # calculate action (P-Lead controller)
364      self.u = self.lineKp * e; # error times Kp
365      # Lead filter
366      self.lineY = (self.u * self.tauZ2pT - self.lineE1 * self.tauZ2mT + self.lineY1 * self.tauP2mT
367      #
368      if self.lineY > 4:
369          self.lineY = 4
370      elif self.lineY < -4:
371          self.lineY = -4
372      # save old values
373      self.lineE1 = self.u;
374      self.lineY1 = self.lineY;
375      # make response
376      par = f"rc {self.velocity:.3f} {self.lineY:.3f} {t.time()}"
377      # debug - no action, go straight
378      #par = f"{self.velocity:.3f} 0 {t.time()}"
379      # debug end
380      service.send("robobot/cmd/ti", par) # send new turn command, maintaining velocity
381      # debug print
382      if True: # self.edge_nUpdCnt % 20 == 0:
383          print(f"% Edge::followLine: ctrl: e={e:.3f}, u={self.u:.3f}, y={self.lineY:.3f}, cnt {self.
384
```

Called on update (10ms), only if in line control

Update interval changed more than 2ms

Get control error

P-controller term

Lead term

Output limit

lineKp = 1.0
lineTauZ = 0.8 (sec)
lineTauP = 0.25 (sec)

Prepare control message

Send to Teensy_interface

Debug print

Software – line sensor

Follow line
Data flow

cmixer.h declaration

```
27  #include "cmotor.h"
28  #include "utime.h"
29  using namespace std;
30
31  /**
32   * The mixer translates linear and rotation reference
33   * values to velocity for each motor.
34   * */
35  class CMixer
36  {
37  public:
38    /** setup and initialize parameters */
39    void setup();
40    /**
41     * Decode messages */
42    bool decode(const char* msg, UTime & msgTime);
43    /**
44     * thread to do updates, when new data is available */
45    void run();
121   /**
122    * Make this visible to the rest of the software */
123   extern CMixer mixer;
```

cmixer.cpp definition

```
36  #include "mvelocity.h"
37  #include "cmixer.h"
38  // create value
39  CMixer mixer;
```

CMixer::decode(msg, time)
{ Decode MQTT message from text to values}
CMixer::run()
{ // Calculate, only if change.

```
198     velDif = wheelbase * turnrate;
199     // adjust each wheel with half difference
200     // positive curvature (CCV) makes the right
201     // wheel turn faster forward
202     v1 = linVel + velDif/2; // right wheel (m/s)
203     v0 = v1 - velDif;       // left wheel (m/s)
```

} // saved to **desiredVelocity** in motor module

lineControl(vel, left, pos
edge.lineControl(
0.25, True, 0.0)

Velocity=0 to disable

Teensy_ interface

Velocity control

Line control

Mixer

Line position

Line detect

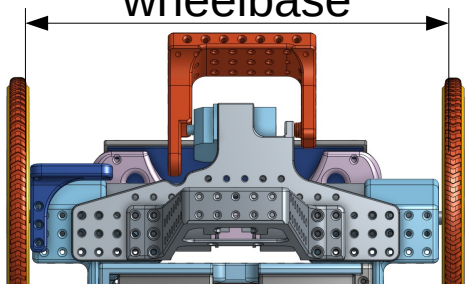
detect

Normalize

Motor

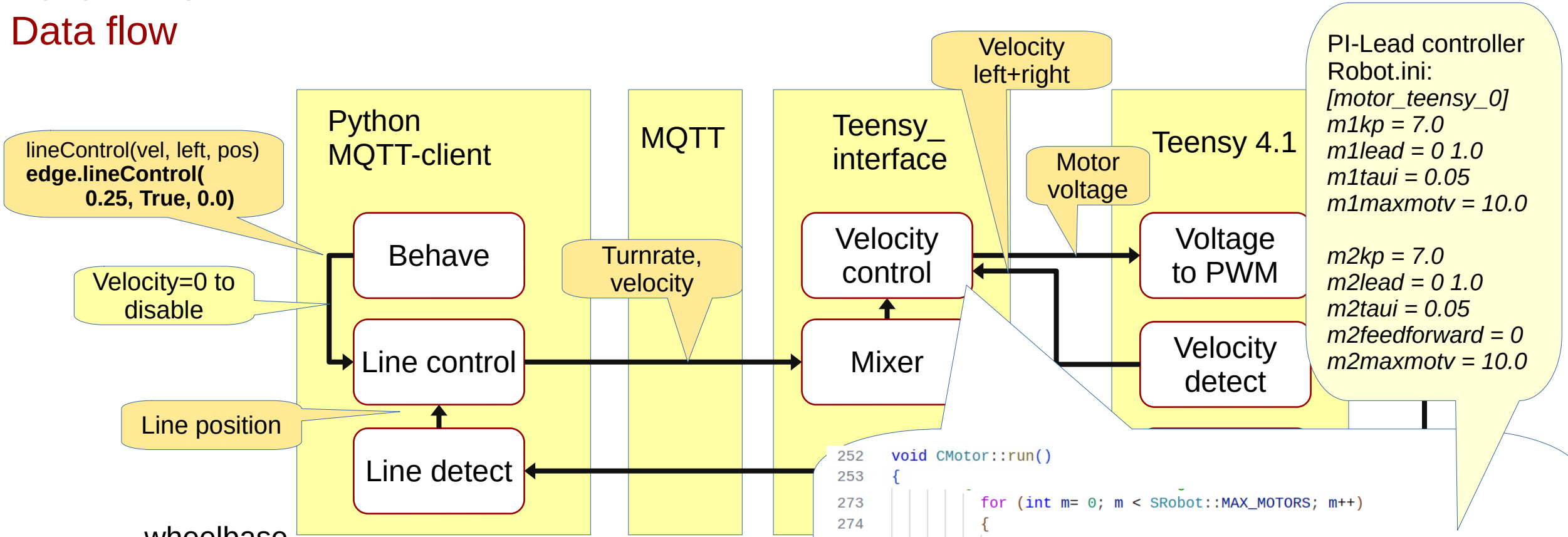
8 x reflectance

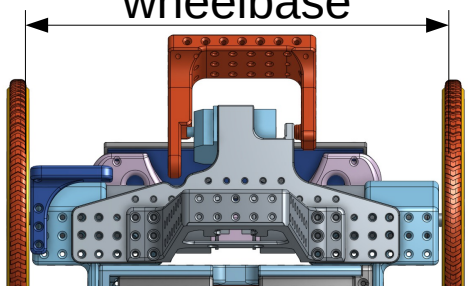
wheelbase



Software – line sensor

Follow line
Data flow





cmotor.cpp

```
252  void CMotor::run()
253  {
273      for (int m= 0; m < SRobot::MAX_MOTORS; m++)
274      {
275          u[m] = pid[m].pid(desiredVelocity[m], mvel[tn].motorVel[m], limit
276      }

287      /// Note that left and right requires different sign to move forward.
288      /// This may be hidden by a sign change in the motor driver firmware.
289      snprintf(s, MSL, "motv %.2f %.2f\n", u[0], u[1]);
290      t.now();
291      teensy[tn].send(s, true);
```

34755 Building dependable robots

- Today
  - Course status
    - Questions before we start
    - Group forming – end today
  - Software intro
    - Line sensor
    - Follow line
  - Group work
    - exercise, instructions in Learn
  - Solve the quiz of the day