```
import sys
import signal
import ctypes
import math
from datetime import datetime
import pixy
from pololu_drv8835_rpi import motors

##### defining PixyCam sensory variables
PIXY_MIN_X = 0
PIXY_MAX_X = 319
PIXY_MIN_Y = 0
PIXY_MAX_Y = 199

PIXY_X_CENTER = ((PIXY_MAX_X-PIXY_MIN_X) / 2)
PIXY_Y_CENTER = ((PIXY_MAX_Y-PIXY_MIN_Y) / 2)
PIXY_RCS_MIN_POS = 0
PIXY_RCS_MAX_POS = 1000
PIXY_RCS_CENTER_POS = ((PIXY_RCS_MAX_POS-PIXY_RCS_MIN_POS) / 2)
BLOCK_BUFFER_SIZE = 10

##### defining PixyCam motor variables
PIXY_RCS_PAN_CHANNEL = 0
PIXY_RCS_TILT_CHANNEL = 1

PAN_PROPORTIONAL_GAIN = 400
PAN_DERIVATIVE_GAIN = 300
TILT_PROPORTIONAL_GAIN = 500
TILT_DERIVATIVE_GAIN = 400

MAX_MOTOR_SPEED = 480
MIN_MOTOR_SPEED = -480

run_flag = 1

# 20ms time interval for 50Hz
dt = 20
# check timeout dt*3
timeout = 0.5
currentTime = datetime.now()
lastTime = datetime.now()
lastLoopTime = datetime.now()


#### defining motor function variables
# 5% drive is deadband
deadband = 0.05 * MAX_MOTOR_SPEED
# totalDrive is the total power available
totalDrive = MAX_MOTOR_SPEED
# throttle is how much of the totalDrive to use [0~1]
throttle = 0
# differential drive level [0~1]
diffDrive = 0
# this is the drive level allocated for steering [0~1] dynamically modulate
diffDrive = 0
# this is the gain for scaling diffDrive
diffGain = 1
# this ratio determines the steering [-1~1]
bias = 0
# this ratio determines the drive direction and magnitude [-1~1]
advance = 0
# this gain currently modulates the forward drive enhancement
```

sensory size of camera

center of sensory field

x position (width) camera only moves horizontally

+ pixy_rcs_center_pos < 0 bad

→ set camera to center

? Sam in bug the camera was set to 0,

parameters of camera motor

PD controller for camera

Jin didn't change

why tilt ?

frame rate of camera 20 ms/frame = 50 Hz

→ inspect image, if nothing print time

Jin didn't Δ

+ to turn, diffdrive should be low, sharper turn, lower diffdrive

how much power going into forward mvmt

proportional drive = pd

```
driveGain = 1
# body turning p-gain
h_pgain = 0.7
# body turning d-gain
h_dgain = 0.2

#### defining state estimation variables
# pixyViewV = 47
# pixyViewH = 75
# pixyImgV = 400
# pixyImgH = 640
# pixel to visual angle conversion factor (only rough approximation) (pixyViewV/pixyImgV +
pixyViewH/pixyImgH) / 2
pix2ang_factor = 0.117
# reference object one is the pink earplug (~12mm wide)
refSize1 = 12
# reference object two is side post (~50mm tall)
refSize2 = 50
# this is the distance estimation of an object
objectDist = 0
# this is some desired distance to keep (mm)
targetDist = 100
# reference distance; some fix distance to compare the object distance with
refDist = 400

panError = 0

turnError_prev = 0

turnError = 0

blocks = None

#### tune parameter
Line1MinYPos = 110;
Line1MaxHeight = 50;
Line2MinYpos = 90;

#means a curve and slow down
paneErrorThreshold1 = 30;
paneErrorThreshold2 = 150;

def handle_SIGINT(sig, frame):
    """
    Handle CTRL-C quit by setting run flag to false
    This will break out of main loop and let you close
    pixy gracefully
    """
    global run_flag
    run_flag = False

class Blocks(ctypes.Structure):
    """
    Block structure for use with getting blocks from
    pixy.get_blocks()
    """
    _fields_ = [
        ("type", ctypes.c_uint),
        ("signature", ctypes.c_uint),
        ("x", ctypes.c_uint),
        ("y", ctypes.c_uint),
        ("width", ctypes.c_uint),
```

(PD control for motor) Drive function: PD page)

Δ from 0.5 → 0.7 final = 0.7 first = 0.5

Δ from 0 → 0.2

in code but not used for ours → earplug stuff

offset from center of pixy sensory field

?

?

maxY position

Jin added in to identify blocks, look only for block on bottom of sensory field

block size

~~max~~ miny position

now offset from center of chip

max pan error, slightly lower

can't make charp turn larger

< 30 straight

30-150 sharp turn

```python
        ("height", ctypes.c_uint),
        ("angle", ctypes.c_uint)
    ]

class ServoLoop(object):
    """
    Loop to set pixy pan position
    """
    def __init__(self, pgain, dgain):
        self.m_pos = PIXY_RCS_CENTER_POS
        self.m_prevError = 0x80000000L
        self.m_pgain = pgain
        self.m_dgain = dgain

    def update(self, error):
        if self.m_prevError != 0x80000000:
            vel = (error * self.m_pgain + (error - self.m_prevError) * self.m_dgain) >> 10
            self.m_pos += vel
            if self.m_pos > PIXY_RCS_MAX_POS:
                self.m_pos = PIXY_RCS_MAX_POS
            elif self.m_pos < PIXY_RCS_MIN_POS:
                self.m_pos = PIXY_RCS_MIN_POS
        self.m_prevError = error

# define pan loop
panLoop = ServoLoop(300, 500)
```

instance of class ServoLoop

```python
def setup():
    """
    One time setup. Inialize pixy and set sigint handler
    """
    global blocks
    pixy_init_status = pixy.pixy_init()
    if pixy_init_status != 0:
        print 'Error: pixy_init() [%d] ' % pixy_init_status
        pixy.pixy_error(pixy_init_status)
        return
    else:
        print "Pixy setup OK"
    blocks = pixy.BlockArray(BLOCK_BUFFER_SIZE)
    signal.signal(signal.SIGINT, handle_SIGINT)
```

Jin = orange comments

not changed

blocksize ↓ signature # ↑ than color

```python
def loop():
    """
    Main loop. Gets blocks from pixy, analyzes target location,
    chooses action for robot and sends instruction to motors
    """
    global blocks, throttle, diffDrive, diffGain, bias, advance, turnError, currentTime,
    lastTime, objectDist, distError, panError_prev, distError_prev, turnError_prev
    global Line1MinYPos, Line1MaxWidth, Line1MaxHeight, Line2MinYpos, paneErrorThreshold1,
    lastLoopTime
    currentTime = datetime.now()
    # If no new blocks, don't do anything
    while not pixy.pixy_blocks_are_new() and run_flag:
        pass
    count = pixy.pixy_get_blocks(BLOCK_BUFFER_SIZE, blocks)
    # If negative blocks, something went wrong
    if count < 0:
        print 'Error: pixy_get_blocks() [%d] ' % count
        pixy.pixy_error(count)
        #sys.exit(1)
```

Jin revised

in file ppt ~~possible~~ we set

Checks to see pixy find new blocks (diff signature sets (color parameters)

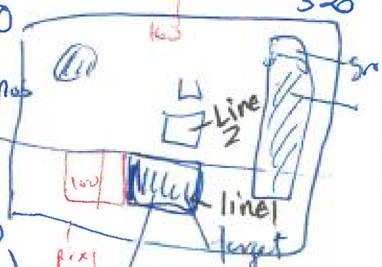
program always receive data from pixy once find something pixyblock = true

∴ loop ~50 Hz (every 20 msec pixy transmit

frequency of loop is up to 50 Hz

- info from Camera
- movement: Pixy back/forth
- update control for motor

```python
    # if more than one block
    # Check which the largest block's signature and either do target chasing or
    # line following
    if count > 0:
        lastTime = currentTime
        # if Pixy sees a guideline, perform line following algorithm
        findBlock1 = 0
        for index in range(0, count):
            if blocks[index].y>Line1MinYPos and blocks[index].height<Line1MaxHeight and
            blocks[index].height<blocks[index].width:
                block1 = blocks[index]
                findBlock1 = 1
                break

        if findBlock1 == 1:
            panError = PIXY_X_CENTER-block1.x

            throttle = 1.0
            diffDrive = abs(float(panError))/300+0.4
            #diffDrive = abs(float(panError))/PIXY_X_CENTER
            advance = 1.0
            panLoop.update(panError)

        #if no block1 is found, just pass
        #else:

    # Update pixy's pan position
    pixy.pixy_rcs_set_position(PIXY_RCS_PAN_CHANNEL, panLoop.m_pos)

    # if Pixy sees nothing recognizable, don't move.
    time_difference = currentTime - lastTime

    if time_difference.total_seconds() >= timeout:
        throttle = 0.0
        diffDrive = 1

    time_difference = currentTime - lastLoopTime
    dt = time_difference.total_seconds()

    # this is turning to left
    if panLoop.m_pos > PIXY_RCS_CENTER_POS:
        # should be still int32_t
        turnError = panLoop.m_pos - PIXY_RCS_CENTER_POS

        # <0 is turning left; currently only p-control is implemented
        ud = h_dgain * float(turnError - turnError_prev)/dt
        bias = - float(turnError + ud) / float(PIXY_RCS_CENTER_POS) * h_pgain
    # this is turning to right
    elif panLoop.m_pos < PIXY_RCS_CENTER_POS:
        # should be still int32_t
        turnError = PIXY_RCS_CENTER_POS - panLoop.m_pos
        # >0 is turning left; currently only p-control is implemented
        ud = h_dgain * float(turnError - turnError_prev)/dt
        bias = float(turnError + ud) / float(PIXY_RCS_CENTER_POS) * h_pgain

    turnError_prev = turnError
    drive()
    lastLoopTime = currentTime
    return run_flag
```



- original code

```
def drive():
    # synDrive is the drive level for going forward or backward (for both wheels)
    synDrive = advance * (1 - diffDrive) * throttle * totalDrive
    leftDiff = bias * diffDrive * throttle * totalDrive
    rightDiff = -bias * diffDrive * throttle * totalDrive

    # construct the drive levels
    LDrive = (synDrive + leftDiff)
    RDrive = (synDrive + rightDiff)

    # Make sure that it is outside dead band and less than the max
    if LDrive > deadband:
        if LDrive > MAX_MOTOR_SPEED:
            LDrive = MAX_MOTOR_SPEED
    elif LDrive < -deadband:
        if LDrive < -MAX_MOTOR_SPEED:
            LDrive = -MAX_MOTOR_SPEED
    else:
        LDrive = 0

    if RDrive > deadband:
        if RDrive > MAX_MOTOR_SPEED:
            RDrive = MAX_MOTOR_SPEED
    elif RDrive < -deadband:
        if RDrive < -MAX_MOTOR_SPEED:
            RDrive = -MAX_MOTOR_SPEED
    else:
        RDrive = 0

    # Actually Set the motors
    motors.setSpeeds(int(LDrive), int(RDrive))

if __name__ == '__main__':
    setup()
    while True:
        ok = loop()
        if not ok:
            break
    pixy.pixy_close()
    motors.setSpeeds(0, 0)
    print "Robot Shutdown Completed"
```

(this is where the program starts) after "sudo python pixy_racer.py"

setup() — function pg 3

ok = loop() function on pg 3 (most important part of code) pixy ~50 frames/sec runs Hz

g's

Line2MinYPos

paneErrorThreshold

kd or kp ?

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\,d\tau + k_d \frac{de(t)}{dt}$$

- $u(t)$: power sent to motor ← controller output
- $k_p e(t)$: systematic error
- $k_i \int_0^t e(\tau)\,d\tau$: ↳ slow progress like temp ctl
- $k_d \frac{de(t)}{dt}$: ↳ rapid Δ like motor ctl

k_p (h_pgain) $e(t)$ (tarnerror)

↳ constant

how quickly you reach set point

too large → overshoot

kd = h_dgain