**ALBiR unit 6: sensorimotor coordination**

**Learning objectives:**

Coordinating sensory signals to modulate on-going motor control is one of the biggest challenges in animal locomotion and mobile robots. What sensory information is important for the task at hand? When to obtain this information? How to transform this information into a control signal for the motor system? In this unit, we will build on what we learned from the previous two units and tackle a classic problem for mobile robots: obstacle avoidance.

**In-class activities:**

**1.** **Lecture in target interception**

Visual tracking is the basis of guidance and navigation. In this lecture, we will introduce some background on target interception as an extension of visual tracking. This will serve as the introduction to the individual simulation report.

**2. Lecture in obstacle avoidance**

Following the lecture on target interception, we will introduce obstacle avoidance in the animals and a simple framework to think about this problem. This will serve as the background for the group coursework of implementing obstacle avoidance.

**3. Implement lane following and obstacle detection [study-group]**

Each robot must be able to drive on a straight lane using visual feedback and detect pre-designed obstacles. You should make use of this study group really to get this behaviour working otherwise the entire team assessment on March 9th could go very wrong. **Do not underestimate implementing behaviours in hardware and start as early as possible.**

1. Perform simple lane-following (assessed on Wednesday 3rd)
2. Detect obstacle: stop robot when the obstacle is in view
3. Compute obstacle distance: stop robot at designated distance from the obstacle in view
4. Avoid stationary obstacles: drive around the obstacles without making contact

Homework 1: Implement obstacle avoidance [group coursework with **live assessment**]

The following tasks will be assessed live with time limit on **March 9th between 9am and 11am**. Each group will be scheduled into a 15min slots for the assessment. Each robot must drive on the straight track given out in the kit.

|  |  |  |
| --- | --- | --- |
|  | **Criteria** | **pt** |
| Task 1 | **Detection of obstacle distance and orientation:** successfully stop the robot at given distance(15~50cm) and angular position(-50° ~ +50°) to the obstacle in two separate trials. We will calculate partial credits for runs where the robot at least detects the obstacle and stops.   * Robot should have both wheels in lane and not pressing on the line * Distance error within 4 cm * Angular error within 10 degrees | 5% |
| Task 2 | **Avoid two stationary obstacles** on the linear track in two separate trials. We will calculate partial credits for runs where the robot avoids at least one obstacle (robot must stay on track before that) and does not stop/freeze.   * Robot should have at least one wheel in lane and not pressing on the line * Robot should not touch the two obstacles * Robot can handle variations of the initial conditions | 5% |

**Game rules:**

* Each team has **10min preparation time** on site for tuning the PixyCam at the assessment location and inputting the given obstacle distance/orientation test values. The team can choose to use a PixyCam settings that the teaching team provides, with penalties.
* Each team can use any robot hardware from the team members but must choose the demonstration platform by the end of the preparation time.
* Each team has at least 5 robots and they serve as redundancy for any team exercises and assessments. Having not working platform will not be a proper mitigation excuse.
* Each team has **5min assessment time** to achieve the maximum performance from the two tasks. Practice any robot operation routine you need ahead.
* Out of lane is defined as one wheel completely outside either of the side lane lines. Stepping on the lane line is ok.
* **You will not be able to use college WiFi**, as IP addresses are hidden. We **HIGHLY** recommend pairing the robot on your mobile hotspot for remote activation purpose. Make sure at least two of the team robots have pre-paired WiFi.
* The assessment condition can be recreated from the printed track and obstacle paper included in the robot kits. The obstacle paper should be folded in half and rolled up to the black line before being taped in a cylinder for the correct assembly.
* You are expected to program and **test your robot at home prior to the assessment**. Do not use the assessment preparation time for coding other than slight tweak of parameters and inserting setpoint values provided at the assessment.
* Videos cannot substitute for live assessments, but we certainly encourage you to shoot videos for your record and troubleshooting purposes. The GTA may request your code for verification (if needed).

Health and safety regulations will be distributed separately on **BlackBoard** for the in-person sessions and live assessments. Guidelines in the following pages will introduce some lower-level code we provide. You are expected to now understand the given codes fully and introduce your own code to implement behaviours.

**Python code provided**

The PixyCam class in PixyCam.py has some methods and attributes to help you.

* newBlocks and oldBlocks stores the information received from the PixyCam’s current and previous frame. Each ‘block’ (element) in these lists has the following attributes: m\_signature (the color ID of the block detected), m\_x & m\_y (the x- and y- pixel coordinates of the centroid of the block detected, measured from the top left corner), and m\_width & m\_height (the height and width of the block detected in pixels). The **blocks are already sorted by pixel size** (largest to smallest).
* getLatestBlocks() gets a new frame from the PixyCam, and updates all of the above variables
* blocksAreNew() checks if the latest frame’s blocks are different in any way to the previous frame’s blocks
* isBiggestSig() returns a 1 if the largest detected block is of the colour that you query, and 0 otherwise
* isInView() returns the index along newBlocks that corresponds to the largest block of the colour that you query, and 0 if that colour was not detected

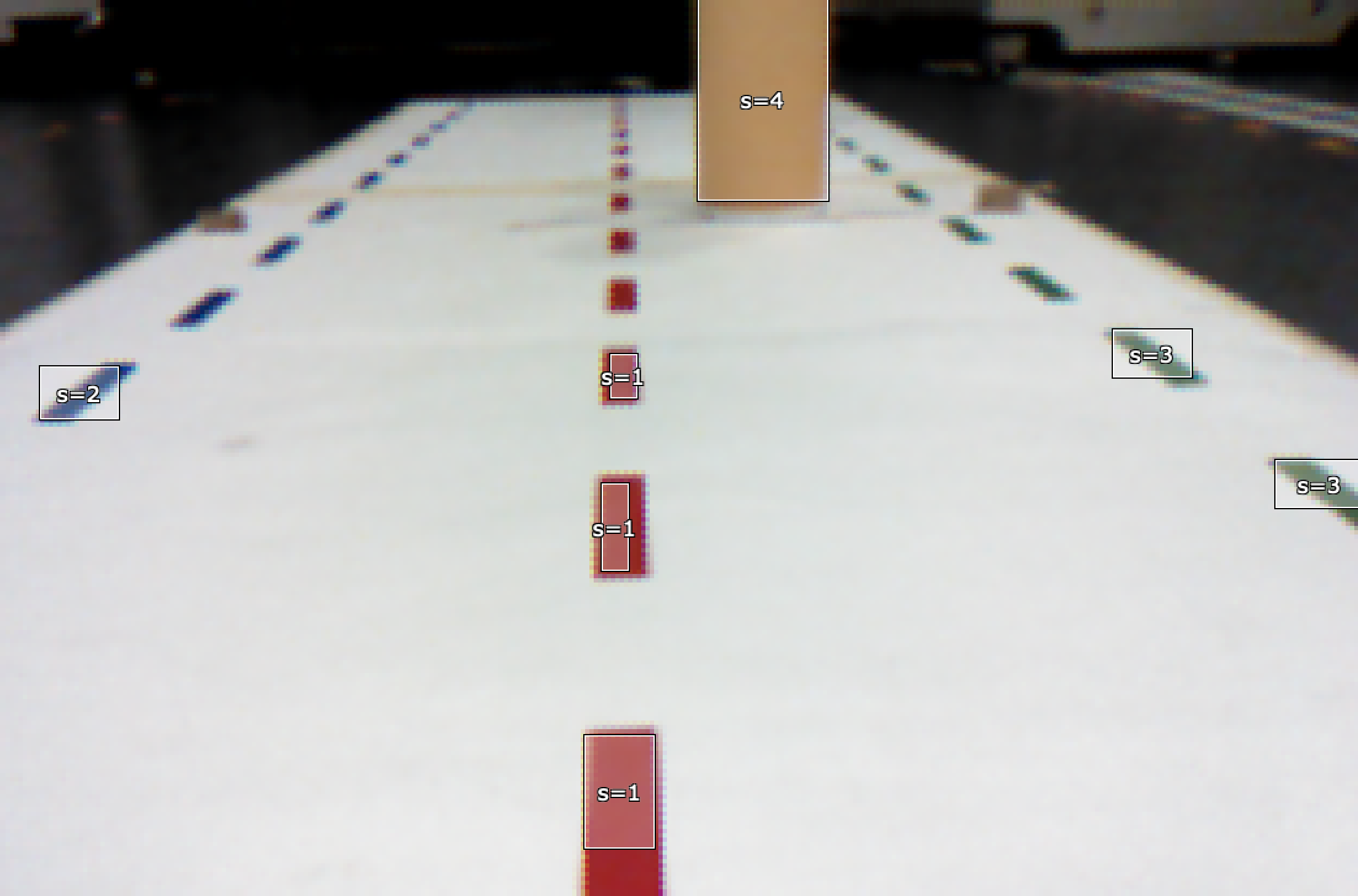
The obstacle avoidance class in the obstacleAvoidance.py has some methods and attributes to help you.

* drive() uses the motor models that from before **to differentially drive the robot**. The drive argument (-1 - +1) is the percentage of the maximum motor speed and bias (-1 - +1) determines how much the robot should turn (negative -> left, positive -> right)
* getBlockParams() takes the index of a block along newBlocks and saves that blocks current size and angle error from PixyCam heading as well as the current frame time in the lists blockSize, blockAngle and frameTimes.
* visTrack() converts the block pixel error into angular error relative to the PixyCam heading and sends command to the servo gimbal. It will allow visual tracking of the identify object via the PID\_controller defined before.

You may need to re-teach the colours to pixy. The centre line should be ID 1, the left line should be ID 2, the right line should be ID 3, and the obstacles should be ID 4. **Please use PixyMon app to confirm the colour tuning before you proceed with any exercises.**

**From last week: basic lane following**

A typical road has lane markers on both sides and in the centre. Your robot has at least these visual cues to stay on the road. A stationary obstacle may be in your way. We have provided the printed track and you can roll up the yellow paper included in the robot kit to make obstacles.



The simplest lane following implementation is to steer the robot toward the largest (nearest) centre land marker. As the nearest marker goes under the robot, the next nearest marker will become the guide. Visual signals tend to be noisy. If you take the target pixel information directly for motor control, the robot will not drive smoothly. What are some methods to avoid such issue? Recall your motor-control and visual tracking exercises and discuss with your teammates. Have you accomplished the group homework on this basic behaviour last week (i.e., laneFollower.py)?

**Level 1: Detect obstacle**

In the classical sense, an obstacle is defined generally as some object in the path. To simplify your task, we have colour-labelled the obstacle with yellow which PixyCam can identify. Please follow the format of the lane following code (i.e. laneFollower.py) and identify the colour signature associated with the obstacle. Insert a conditional statement to stop the robot when PixyCam sees an obstacle. You may keep your pan servo locked to centre for now. To start, take the completed follow() function in the laneFollower.py and place in stopAtStationaryObstacles() function in the obstacleAvoidance.py. Complete level 1 in the stopAtStationaryObstacles() function in the obstacleAvoidance.py.

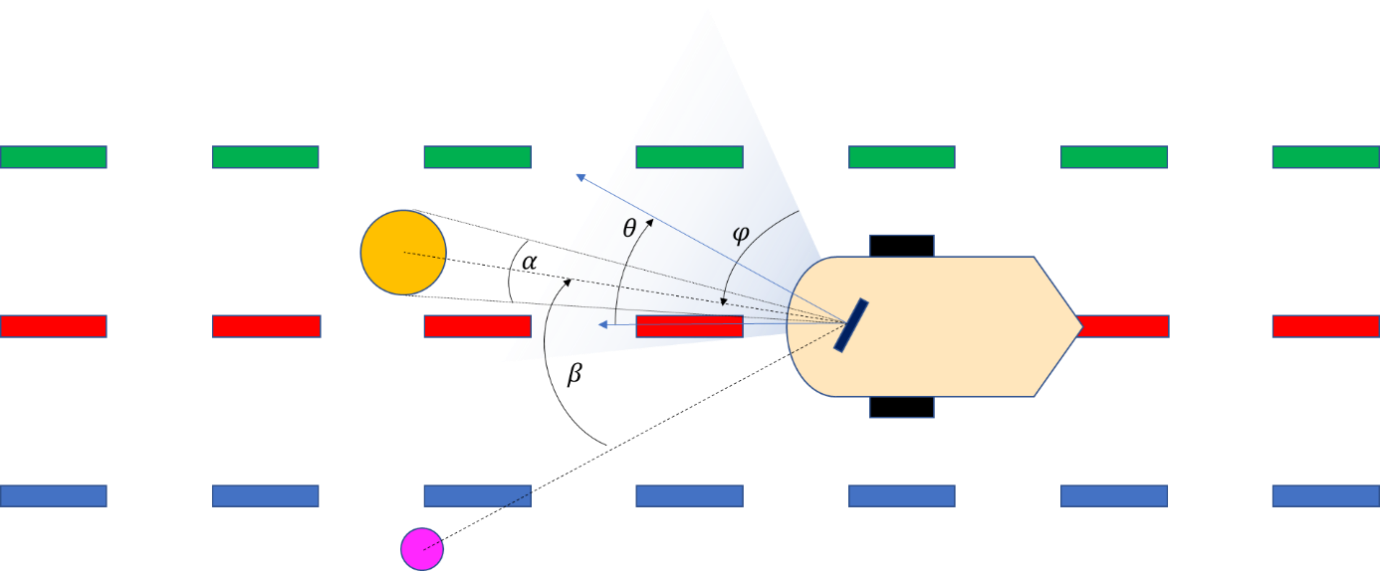
**Level 2: Cue obstacle distance**

Now that you have identified the obstacle, you need to write a small function to estimate its distance from the robot. We have introduced various biological strategies in the lecture. However, for simplicity the object elevation on a flat surface might be the easiest to implement. Once you have a good distance estimation, modify your conditional statement to stop the robot at any given distance between 15cm and 50cm. You should be able to stop the robot within 4cm deviation.

Complete level 2 in the stopAtStationaryObstacles() function in the obstacleAvoidance.py.

**Level 3: Cue obstacle distance & orientation (assessment Task 1)**

Now, modify your conditional statement from Level 2 and stop the robot when the obstacle is within a given distance and at **a specific angular position relative to the robot’s heading**. You should be able to achieve the angular accuracy within 10 degrees. For the purpose of this assessment, it is legal to perform spot-turn after stopping at the right distance to achieve the angular position requirement.



This schematic shows the general conditions of your **robot driving environment in the final robot race**. The lane markers are labelled by three different colours. You may also see an obstacle (in yellow) and an occasional landmark in the road. More details on the robot race will be given in the next class.

Complete level 3 in the stopAtStationaryObstacles() function in the obstacleAvoidance.py. The stopping obstacle distance (15~50cm) and angular position (-50° ~ +50°) values will be given at the assessment.

**Level 4: Avoiding stationary obstacles (assessment Task 2)**

An obstacle is defined as an object that is on the path of your robot. From the basic lane following behaviour, we have now defined a “path” that the robot should follow, thus the definition of an obstacle becomes clear. In classical robotics, path-planning often involves a global reference frame in which both the obstacle parameters and the robot states are defined. For animals, such a global frame is often not available or not used. Instead, a model of the motion is mapped directly to some sensory parameter space. Using geometry, please come up with a model of what a stationary obstacle would “look like” to the robot if it were in and out of the way of the robot path. Recall biological and bio-inspired target interception and obstacle avoidance strategies covered in the lectures. Can you make use of any ideas to avoid the obstacle?

Leverage what you have done in previous Levels, try to steer the robot away from the obstacle as needed while still staying on the track. Complete the avoidStationaryObstacles() function in the obstacleAvoidance.py script to make the robot follow the straight lane test track while avoiding two consecutive stationary obstacles on the path. During the study group, the GTA will demonstrate how the obstacle track will be set up so you can plan accordingly.