Robotics Practical Assignment – Predator & Prey

**Technical Design**

**Player Behaviour – Predator (Search & Destroy)**

The ‘Predator’ mode applies computer vision techniques using the Turtlebot’s exteroceptive sensor data and, based on this data, utilises its effectors and actuators to ‘search and destroy’ its prey. When activated, the predator bot will search its surroundings for its target using the provided ‘green hat’ as sample data. Upon detection of the other bot, it will pursue it until it reaches a pre-defined safe proximity (to avoid damage to either bot) and notify the user that it has caught the prey through a distinctive action. Should obstacles be encountered or introduced, the bot uses its sensor data to determine the best avoidance action to take to circumvent them before resuming its search or pursuit.

**Opponent Detection**

**Identification**

The provided ‘green hat’ the Turtlebots would wear was sampled in different lighting conditions to identify a robust set of colour value ranges, which could then be used to segment it via colour slicing. Due to its inherent performance benefits and its closer representation of real-world colour, the HSV colour space was used to define these ranges. Each channel has hard-coded upper and lower values taken from unaltered screenshots of the Kinect camera’s view of the hat. Creating a binary mask with these ranges enabled the Turtlebot to identify its opponent when it was in the camera’s field of view.

SCREENSHOT: GREEN HAT (RAW KINECT VS BINARY MASK)

**Location**

Implementing a ‘search’ function was necessary to detect the opponent if it started out of the predator’s view or moved beyond the range of its Kinect camera. Using the previously constructed binary mask, conditional logic was set that would initiate a “search\_for\_prey” function if the mask’s largest singular object – the green hat – contained less than a certain number of pixels. This function then constructs and publishes an object of the ‘Twist’ class to the Turtlebot’s “cmd\_vel” topic, setting the ‘z’ property of its angular velocity to a sensible speed to make it rotate on the spot. The rotation lasts…

In doing so, the Turtlebot has now been able to see all unobstructed areas in its general vicinity and – assuming it detects it – can proceed in pursuing its prey.

Should the basic default search motion fail in locating the prey, it will then…

**Pursuit**

Doing stuff is cool

**Capture**

**Obstacle Avoidance**

**Identification**

The predator needed to differentiate between its prey and any other objects it may find itself near. While the prey was clearly definable by its distinctive green hat the feedback from the Turtlebot’s Kinect camera

**Action**

The

**Player Behaviour – Prey (Run & Hide)**

**Opponent Detection**

The

**Obstacle Avoidance**

The

**Testing & Evaluation**

**Functionality**

Efficient implementation

**Performance**

Accuracy e.g. time/distance

**Limitations**

?

**Possible improvements**

?

**Influence of system parameters**

Linear/Angular velocities, CPU speeds, controller values

References

**There are no sources in the current document.**