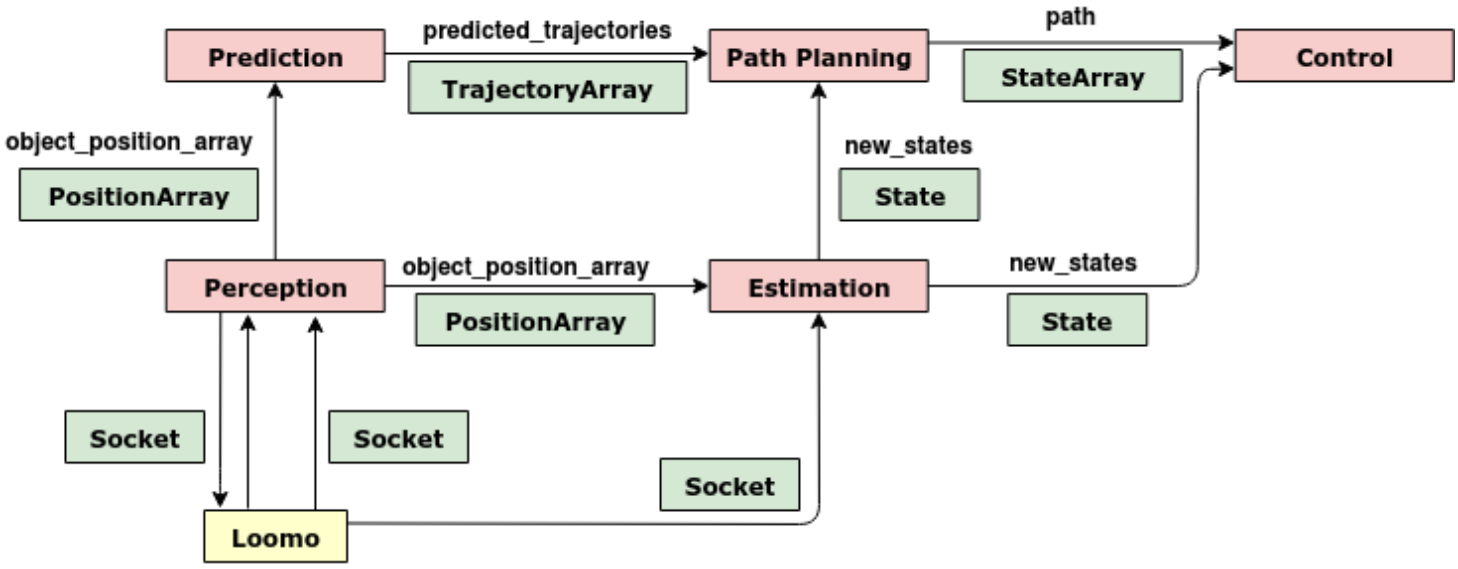
ROS Structure



Perception

1. Detector initialization: Modify parameters if a new detector is set.

```
# Initialize Detector Configuration --> Loomo received image dimensions: 80x60x3
detection_image = DetectorConfiguration(width = w, height = h,
                                        channels = c, downscale = d,
                                        global_path = 'path',
                                        detector = detector_class())
```

2. Receive image from Loomo via Socket.

3. Detector function Requirements:

Variable	Input/Output	Description	Example
opencvImage	Input	List of data_size RGB data	[255, 5, 8, 157, 255, 0, ...]
bbox_list	Output	List of bounding boxes	$[[x_{center}, y_{center}, width, height]_1, ...]$.
label_list	Output	List of labels	$[label_1, ...]$

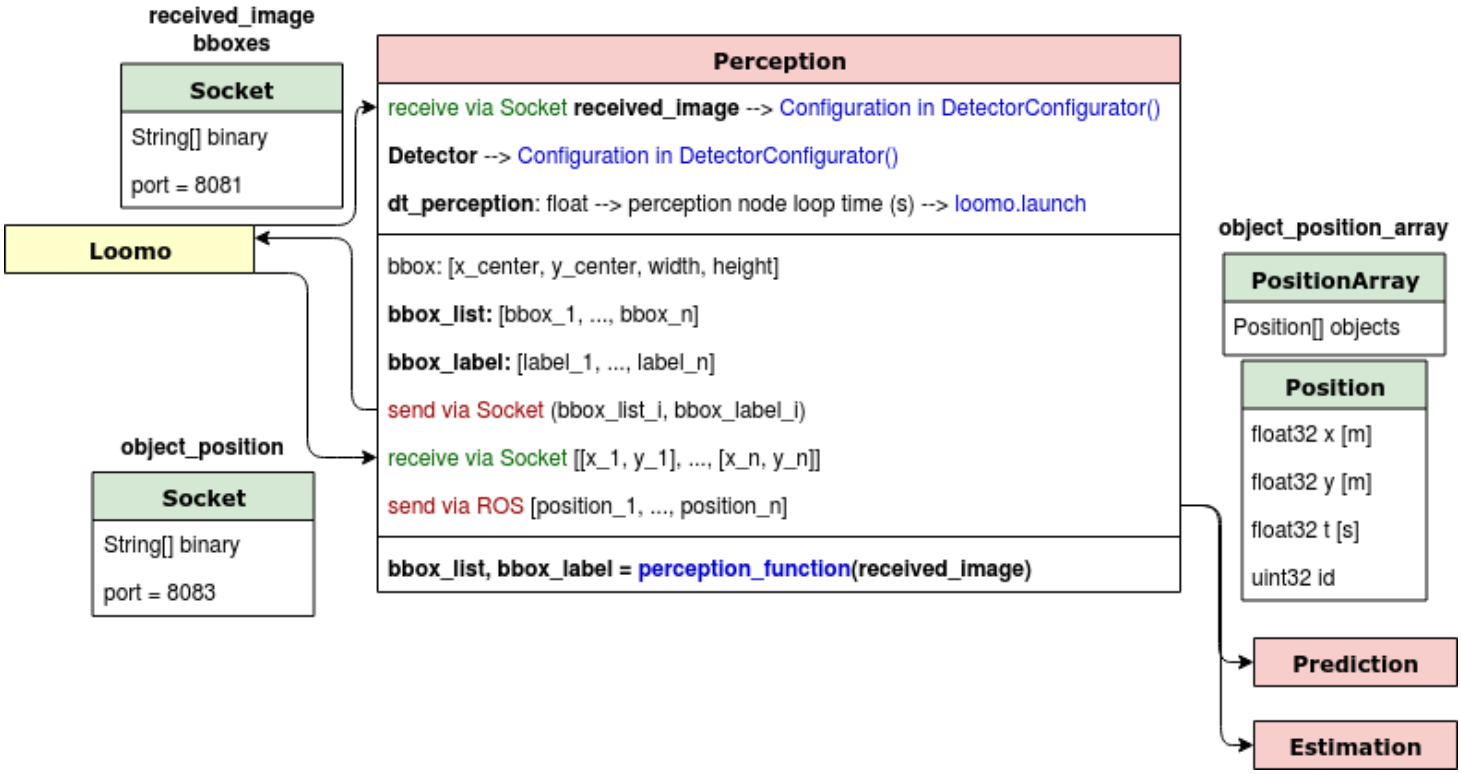
```
def detect(self, received_image):
    pil_image = Image.frombytes('RGB', (width/downscale, height/downscale), received_ima
    opencvImage = cv2.cvtColor(np.array(pil_image), cv2.COLOR_RGB2BGR)
    opencvImage = cv2.cvtColor(opencvImage,cv2.COLOR_BGR2RGB)
    bbox_list, bbox_label = detector.forward(opencvImage)
```

4. perception.launch

Change the IP address and the time period of perception if needed:

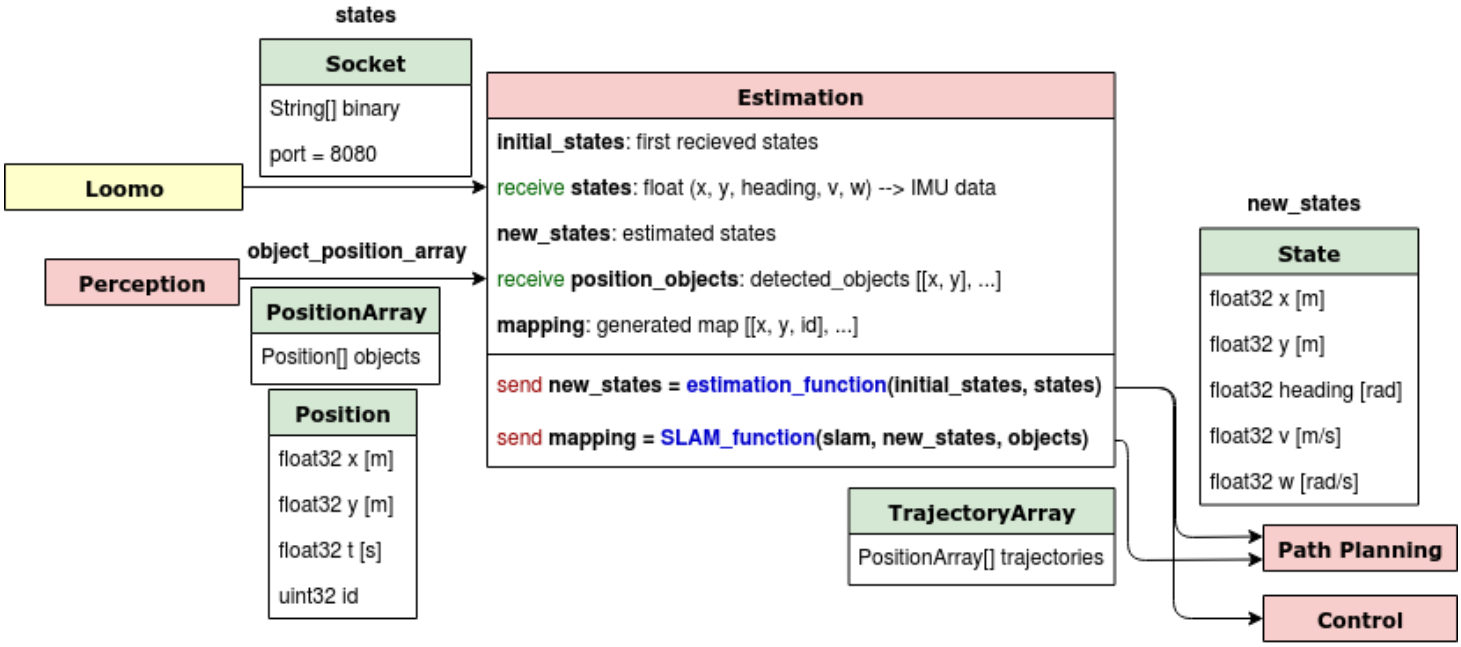
```
<param name="ip_address" value="" />
<param name="dt_perception" value="" />
```

Software Architecture

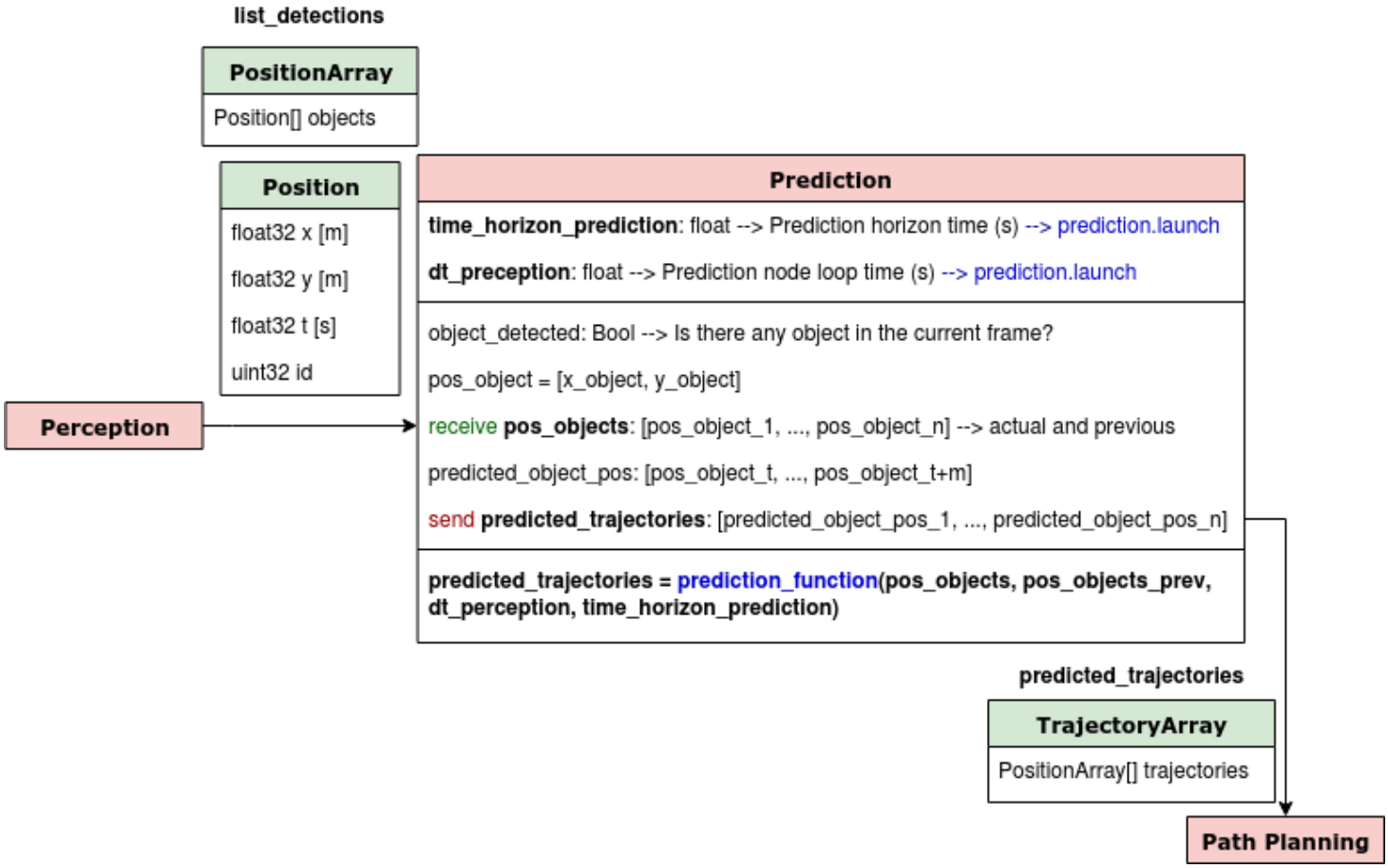


Estimation

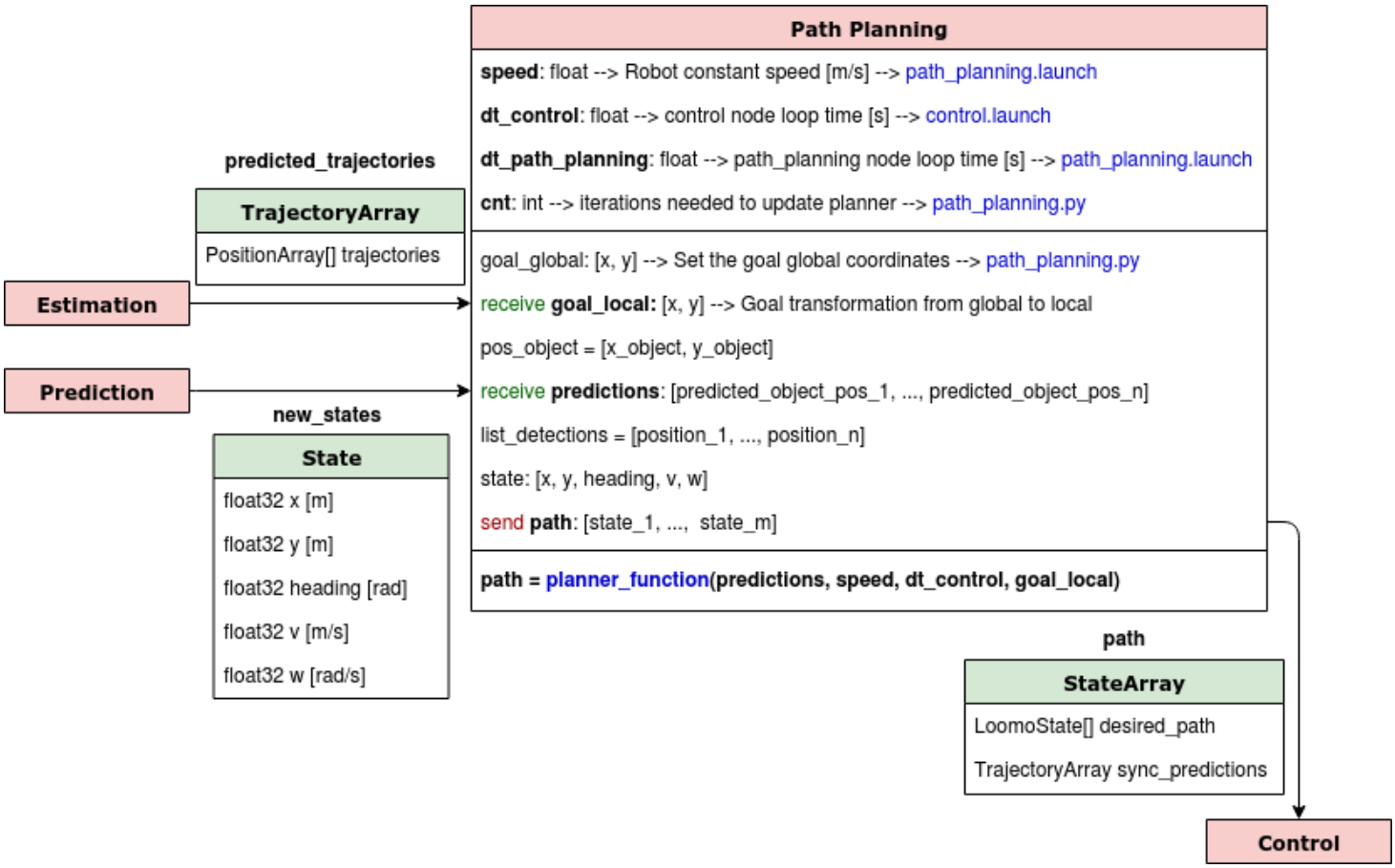
Software Architecture



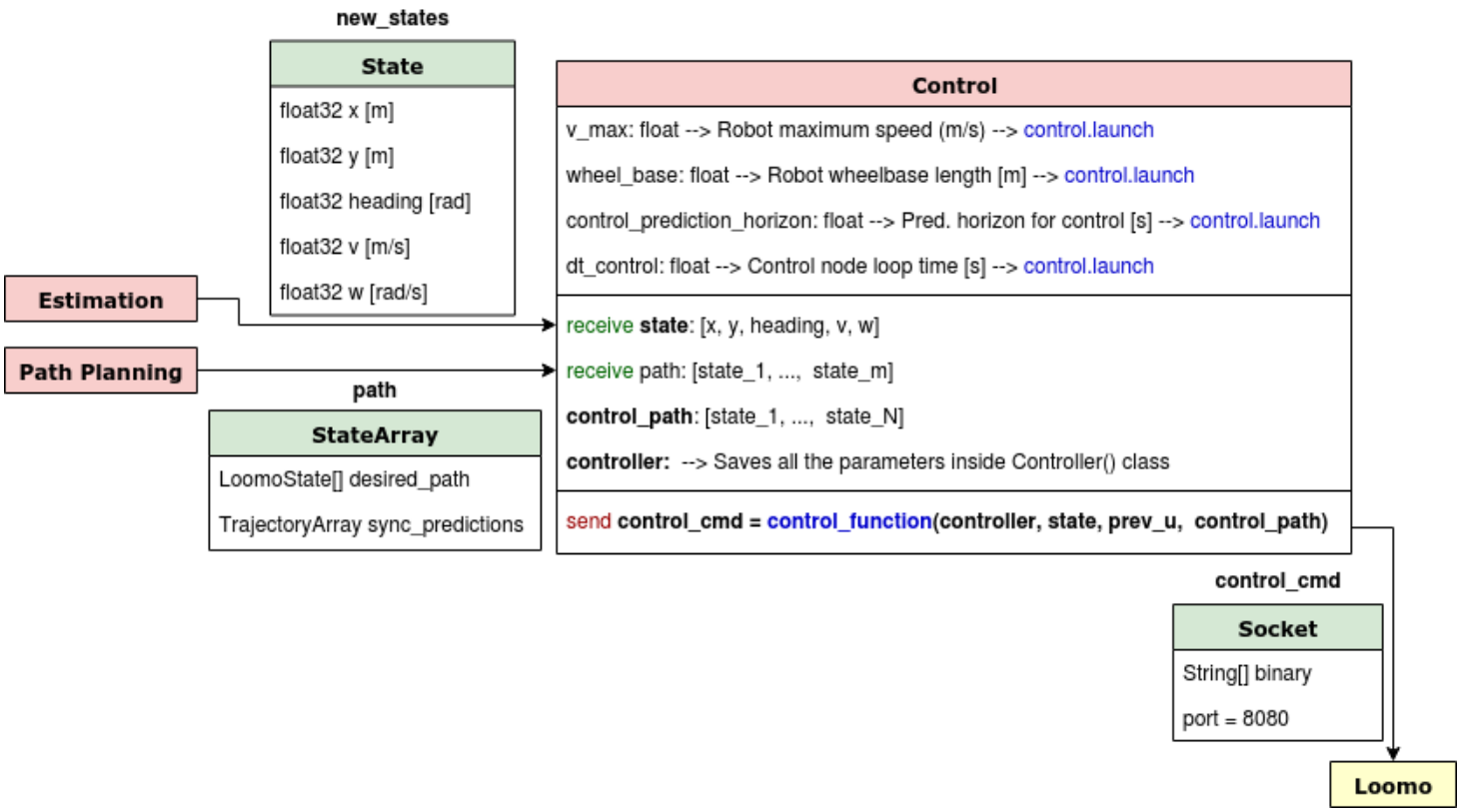
Prediction



Path Planning


Path Planning
speed: float --> Robot constant speed [m/s] --> path_planning.launch
dt_control: float --> control node loop time [s] --> control.launch
dt_path_planning: float --> path_planning node loop time [s] --> path_planning.launch
cnt: int --> iterations needed to update planner --> path_planning.py
goal_global: [x, y] --> Set the goal global coordinates --> path_planning.py
receive goal_local: [x, y] --> Goal transformation from global to local
pos_object = [x_object, y_object]
receive predictions: [predicted_object_pos_1, ..., predicted_object_pos_n]
list_detections = [position_1, ..., position_n]
state: [x, y, heading, v, w]
send path: [state_1, ..., state_m]
path = planner_function(predictions, speed, dt_control, goal_local)
predicted_trajectories
TrajectoryArray
PositionArray[] trajectories
Estimation
Prediction
new_states
State
float32 x [m]
float32 y [m]
float32 heading [rad]
float32 v [m/s]
float32 w [rad/s]
path
StateArray
LoomoState[] desired_path
TrajectoryArray sync_predictions
Control

Control


new_states
State
float32 x [m]
float32 y [m]
float32 heading [rad]
float32 v [m/s]
float32 w [rad/s]
Control
v_max: float --> Robot maximum speed (m/s) --> control.launch
wheel_base: float --> Robot wheelbase length [m] --> control.launch
control_prediction_horizon: float --> Pred. horizon for control [s] --> control.launch
dt_control: float --> Control node loop time [s] --> control.launch
receive state: [x, y, heading, v, w]
receive path: [state_1, ..., state_m]
control_path: [state_1, ..., state_N]
controller: --> Saves all the parameters inside Controller() class
send control_cmd = control_function(controller, state, prev_u, control_path)
Estimation
Path Planning
path
StateArray
LoomoState[] desired_path
TrajectoryArray sync_predictions
control_cmd
Socket
String[] binary
port = 8080
Loomo