National University of Singapore
Department of Mechanical Engineering

Semester 2, AY2023-24

ME5413 Autonomous Robot Navigation

Simon Kenneth

Assignment 1

1 Task1

1.1 Template Matching

Template matching is a common image processing method that attempts to find the best match within a search region given a template image. Various specific implementation methods exist for template matching, such as Square Difference Matching and Cross-Correlation Matching. In this assignment, Normalized Cross-Correlation Matching was adopted because it considers the average intensity of both the image and template, calculating the correlation by subtracting the average. Consequently, it is more robust as it mitigates the effects of brightness and contrast changes. Let T denote the template image and I be the image to be matched, respectively. The Normalized Cross-Correlation Matching can be expressed as:

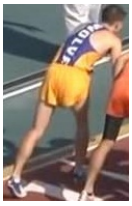
$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}} \tag{1}$$

Where:

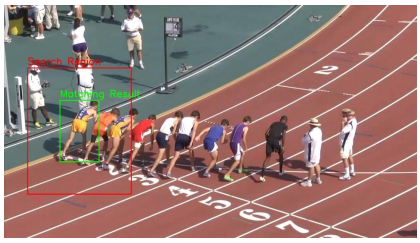
$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$$
$$I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'',y+y'')$$

Here, x, y represent the pixel coordinates and w, h denote the width and height, respectively. This assignment implemented the template matching using OpenCV for Python. Specifically, the function **cv2.matchTemplate()** with parameter **cv2.TM_CCOEFF_NORMED** was adopted. The matching results from **cv2.matchTemplate()** need to be further processed by **cv2.minMaxLoc()** to obtain the best match value and its corresponding location in the search region. The following figure illustrates the best match result for a given image, search region, and the template image.

For task 1, single object tracking (SOT) can be implemented by iteratively applying template matching. Specifically, the initial template image can be obtained from the bounding box data in *firsttrack.txt*. Assuming the search region has been manually specified, when the program reads a new image frame, we can use the image template to find the best matching sub-image in the new image and update the image template using the matching result. By repeating this process, we can obtain all the matching results in the form of coordinate series in an image sequence, which effectively represents the moving trajectory of the object we want to track.



(a) Template Image



(b) Results of Normalized Cross-Correlation Matching.

Figure 1: An example of template matching. The red bounding box indicates the search region, while the green one indicates the matching result.

However, numerous issues arise when employing template matching as the SOT method. One of the most critical problems is that the template itself lacks awareness of the subject it is tracking. It merely mechanically identifies subimages that are as similar as possible based on the template, disregarding the semantic information of the tracked subject. Consequently, when a matched subimage that does not encompass the complete tracked object due to movement or other factors is used as the new template, the algorithm will continue matching with incomplete information in the subsequent iterations, resulting in incomplete matching, even if template matching deems this subimage to have the highest matching value.

Furthermore, the erosion of the tracked object during template matching is irreversible because the template lacks knowledge of ”what is the object.” Another consequence of this issue is that template matching is easily interfered with by static objects in the search region. These objects are more likely to persist in the template as they are not eroded by movement like the tracked objects. Consequently, the template matching will repeatedly employ templates containing these static objects for matching. As a result, the tracked object will consistently erode while the static objects remain in the template until the tracked object is completely lost in the template, indicating a tracking failure.

Another problem with template matching-based SOT is that the template scale is not automatically adjusted, which significantly affects the tracking performance in terms of Intersection over Union (IOU) metric.

To address these issues, various tricks and improvement methods have been proposed in this assignment. Further details will be discussed in section 1.3.

1.2 Kalman Filter

A more general concept underlying the Kalman filter is the Bayesian filter, which utilizes prior knowledge, past estimations, and current observations to update beliefs about the system state. As a special instance of a Bayesian filter, the Kalman filter assumes linearity in both the state transition and measurement (or observation) model of the system. It also assumes that all random variables, such as process noise and observation noise, follow Gaussian distributions, making the Kalman filter more mathematically tractable.

To utilize the Kalman filter, we need state-space equations in the form of differential equations to describe the system's behavior mathematically. Assuming the state vector as $\boldsymbol{x}$ and the control vector as $\boldsymbol{u}$, respectively, the state-space equation can be presented as:

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + \boldsymbol{w} \tag{2}$$

Here, A and B represent the system dynamics matrix and control input matrix, while $\boldsymbol{w}$ signifies white noise following a Gaussian distribution. Occasionally, due to constraints, the complete system state cannot be directly observed. In such instances, we introduce another variable known as the measurement vector $\boldsymbol{z}$, which correlates with the state vector to denote observations such as sensor readings. The relationship between the measurement vector and the state vector can be elucidated by the following system measurement equation:

With the aforementioned terminologies in place, the core ideas of the Kalman filter can be summarized into the five key steps, see Appendix A.

For the object tracking task, our primary concern lies with determining the object's location within the image. In this regard, let's define the state vector used in the Kalman filter as $\boldsymbol{x} = (x, y, v_x, v_y)^T$. This yields the subsequent state transition function as follows:

$$\boldsymbol{x}(k+1) = \begin{pmatrix} x(k+1) \\ y(k+1) \\ v_x(k+1) \\ v_y(k+1) \end{pmatrix} = \begin{pmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x(k) \\ y(k) \\ v_x(k) \\ v_y(k) \end{pmatrix} + \boldsymbol{w}(k) \tag{3}$$

As the moving velocity cannot be directly observed due to template matching yielding only coordinate information, the measurement matrix is defined as $H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$. In each update loop, the predicted state $\hat{\boldsymbol{x}}(k)$ can be obtained from the Kalman filter based on the new observation $\boldsymbol{z}(k) = (x(k), y(k))^T$ from the template matching. The update step *implicitly* estimates the moving speed owing to the state transition function, which aids in predicting the state at the subsequent timestamp.

Thanks to the prediction features provided by the state transition equation in the Kalman filter, the influence of some abnormal measurements can be eliminated by the weighted effect of Kalman gain on observation error. For example, the result of template matching may significantly drift because of static objects at a certain timestamp. In this case, the Kalman filter will deem the observation as unreliable and therefore trust the prediction result more. Macroscopically, this manifests as the Kalman filter aiding the template in overcoming the influence of static objects by predicting the template's movement speed.

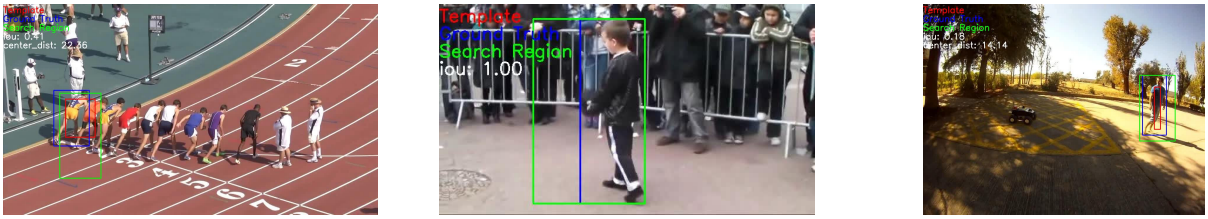
1.3 Improvement

1.3.1 Parameter Tuning

1. Sequence-specific sizes of template and search region.
   In fact, the effectiveness of template matching heavily relies on choosing appropriate sizes for both the template and the search region. A smaller sizes for the search region can mitigate the impact of stationary objects and cluttered data on template matching. However, it may result in incomplete object detection if the object moves too quickly and goes beyond the search area's boundaries. This highlights the need for a larger search area, presenting a trade-off that requires careful consideration.
   Similarly, adjusting the initial sizes of the template may be necessary, as it could contain less essential information about the tracked object. Decreasing the template size adequately can

improve the tracking of significant object features, thus enhancing tracking accuracy, albeit at the expense of IoU.
Since different objects and backgrounds in the provided image sequences have different features, specifying different scales for the search region and template will achieve better tracking results. Figure 2 illustrates the example of sequence-specific adjustments in search region and template sizes.

2. Tune the Kalman filter.
To further improve the object tracking performance using the Kalman filter, it's important to adjust specific parameters of the filter for different sequences.
For different sequences, it's crucial to consider the general motion direction and introduce prior velocity guess of the tracked objects. For example, in sequence one where athletes primarily move towards the upper right of the image, introducing a velocity state prior with $v_x > 0$ and $v_y < 0$ can improve the initial template matching accuracy. This prior aids the Kalman filter in predicting the object's future position more accurately.
The matrices Q and R are also significant in the Kalman filter as they determine the preference between prediction and observation trust. Given the instability of observations in task 1 (i.e., the results of template matching) while the objects maintain a stable velocity, more trust can be placed on the predicted states of the Kalman filter. Specifically, a larger value should be assigned to R compared to other elements in Q, indicating greater measurement noise based on prior knowledge.
Conversely, the components related to speed in Q, i.e., the 3rd and 4th elements on the diagonal, should be adjusted to smaller values. This adjustment encourages the Kalman filter to rely more on speed prediction.



(a) Sequence One (b) Sequence Two (c) Sequence Three

Figure 2: Illustration of sequence-specific sizes of template and search region. The red bounding box indicates the search region, while the green one indicates the template.

The fine parameter adjustments on each sequence pose a significant challenge of data management. In this case, this assignment utilizes a YAML file as the parameter management tool, which greatly facilitates the efficiency of parameter tuning. All the parameters are stored in *config/param.yaml*.

1.3.2 Multi-Scale Template

An inevitable problem in object tracking is that the geometric perspective, or the scale of the tracked object may be constantly changing, which is especially obvious in sequences one and five. The changing scale poses an extra challenge for template matching since the matching score will decrease if the scale of the object in the template is different from that of the search region, even if the color and texture features are the same.

In this case, multi-scale template matching was proposed in this assignment. To be more specific, before starting a new template matching, the template image will be scaled into different sizes, which actually establishes an *image pyramid*. Then, each image template will be used to conduct an individual matching, and the one that has the highest matching value will be selected as the image template for the next step. By doing this, a template can still be well-matched to the corresponding sub-image even if the objects are shrinking or enlarging.

1.3.3 Subject Feature-based Adaptive Template Size

As discussed earlier, a significant issue with template matching-based object tracking is that the template lacks knowledge of the object it should track. To address this, more efficient methods for feature extraction need to be employed either in the image template or the search region.

CNN-based (ResNet-18 was used in this assignment) feature extraction has also been attempted in this task. Unfortunately, the results were unideal. This is because an off-the-shelf CNN model pre-trained on a large dataset, lacks the specificity required to extract the pertinent features for object tracking. One potential enhancement for this approach is to fine-tune the CNN model on a dataset specifically tailored for object tracking. This would instruct the model to extract the relevant features of the moving object while disregarding irrelevant ones. However, such a dataset is currently unavailable for this task.

In this scenario, a simple object feature extraction method based on the HSV color space can be proposed. The Hue, Saturation, and Value (HSV) color space are commonly used for object segmentation in computer vision. HSV separates color information from brightness, enabling robust segmentation. Its intuitive representation aligns well with human perception, and its resilience to lighting variations enhances its suitability for real-world applications.

(a) The original image. (b) The segmented image using HSV space.

Figure 3: An example of HSV space-based object segmentation. The HSV threshold used in this example is $(0, 124, 103)$ for the lower bound and $(11, 255, 255)$ for the upper bound.

As shown in Figure 3, once the HSV threshold is appropriately set, the object of interest can be clearly extracted into white in a binary image, such as the volleyball player wearing red clothing. The opening operation (Corrosion followed by expansion) is adapted followed by the segmentation to get more robuts feature extraction.

Given the initial image template, the corresponding segmented image can be obtained. From that, we can determine the occupancy proportion and the relative position of tracked subjects in the template by rectangle contour extraction. Once the object moves in the image, the relative position and occupancy proportion will change accordingly. In such cases, the template position and size can be adjusted to keep the tracked object in the center of the template and maintain the occupancy proportion so that a better IoU can be achieved.

By doing so, the object's semantics are introduced in template matching as prior knowledge, which significantly reduces the issues of erosion of the tracked objects.

Considering the above insights, an intuitive and promising method can be proposed using segmentation neural networks to further enhance the tracking performance. In fact, this is what the recently published work, namely "Robust visual tracking by segmentation" by ETH Zurich in 2022, did. This assignment has reproduced this paper for the given task, and the tracking results are indeed very impressive. However, since there are currently no new innovative improvements based on that paper, detailed discussion will not be covered here. Please check "*Task 1/results/seg_based_effect.mp4*" for running effect of this method.

2 Task2

2.1 System Model

For task 2, physics model-based trajectory prediction methods were utilized. Assuming access to the vehicle's current position x, y, velocity v_x, v_y, and acceleration a_x, a_y, the forward prediction equation can be expressed as follows:

$$\mathbf{x}(k+1) = \begin{pmatrix} x(k+1) \\ y(k+1) \\ v_x(k+1) \\ v_y(k+1) \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x(k) \\ y(k) \\ v_x(k) \\ v_y(k) \end{pmatrix} + \begin{pmatrix} 0.5\Delta t^2 & 0 \\ 0 & 0.5\Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix} \begin{pmatrix} a_x(k) \\ a_y(k) \end{pmatrix} \quad (4)$$

The constant velocity is just a special instance of the above equation.

The Constant Turn Rate and Velocity(CTRV) model can written as:

$$\mathbf{x}(k+1) = \begin{pmatrix} x(k) + \frac{v(k)}{\omega(k)} \sin(\theta(k) + \omega(k)T) - \frac{v(k)}{\omega(k)} \sin(\theta(k)) \\ y(k) - \frac{v(k)}{\omega(k)} \cos(\theta(k) + \omega(k)T) + \frac{v(k)}{\omega(k)} \cos(\theta(k)) \\ \theta(k) + \omega(k)T \\ v(k) \\ \omega(k) \end{pmatrix} \quad (5)$$

Where $v(k)$ represents the absolute value of the current velocity, and $\omega(k)$ represents the current turning rate.

In this assignment, the current acceleration is derived by calculating the difference in velocity between the current and previous timesteps. Similarly, the current turning rate is determined by the difference in heading angle.

2.2 Analysis

The prediction results of three employed methods for the specified agents concerning ADE and FDE are presented in Table 1.

The unexpected outcome is that the constant velocity-based prediction outperforms constant acceleration-based methods in most cases. To elucidate this phenomenon, the predicted trajectories for some representative agents are visualized in Figure 4. From this, we can observe that

the prediction closely aligns with the numerical results. The trajectories generated by constant acceleration exhibit significant deviation from the ground truth, particularly in the cases of agents 6 and 15. This deviation arises because the current acceleration might not be reliable due to potential frequent acceleration and deceleration, especially in cornering scenarios. Consequently, we can observe that the predicted trajectory for agent 15 even reverses direction, initially accelerating towards the lower left but later accelerating in the opposite direction in reality.

In contrast, trajectory prediction results based on CTRV are closer to the ground truth. The reason is that CTRV takes into account turning speed, so the prediction in turning scenarios will be more reasonable.

3 Bonus Task

The bonus task is essentially a high-level encapsulation based on ROS. A ROS node must be designed to access the published images.

For this task, a Python class was devised, featuring multiple ROS subscribers and publishers as member variables. The subscribers receive images published by ROS bag, and the respective callback function within the class is responsible for storing the latest received images and performing preprocessing tasks, such as converting the data from ROS message format into an OpenCV Mat object. Upon receiving a new image, the subscriber's callback function invokes the object tracking function to determine the bounding box of the moving object. The tracking results are then encapsulated into ROS messages and published to specific topics.

The running effect in Rviz for this task is illustrated in Fig. 5.

To save TA's the time and effort required for configuring ROS-related code, this task foregoes the traditional workspace and ROS package-based code composition structure. Leveraging Python's Compile-free feature, all necessary functions can be implemented within a single Python script, eliminating the need for annoying CMake and package XML files. This simplification is possible because rospy, which facilitates ROS communication, utilizes ROSTCP/ROSUDP as the underlying mechanism. Consequently, a Python script can effortlessly obtain port information of other nodes through roscore/rosmaster using a fixed port (11311) and establish communication without additional code.

As a result, only TWO files are required to execute the additional task. A shell script has also been developed to launch all necessary programs—such as roscore, the core Python code, and rviz—in one go. For detailed instructions on running the program, please refer to *README.md*.

A Five Key Steps of Kalman Filter

- Predict

  1. Calculate the prior state estimation.

$$\hat{\boldsymbol{x}}'(k) = A\hat{\boldsymbol{x}}(k-1) + B\boldsymbol{u}(k-1) \tag{6}$$

  2. Calculate the covariance matrix of prior state estimation error.

$$P'(k) = AP(k-1)A^T + Q \tag{7}$$

- Update

  3. Calculate Kalman gain.

$$K(k) = \frac{P'(k)H^T}{HP'(k)H^T + R} \tag{8}$$

  4. Calculate the posterior state estimate.

$$\boldsymbol{x}(k) = \boldsymbol{x}'(k) + K(k)(\boldsymbol{z}(k) - H\boldsymbol{x}'(k)) \tag{9}$$

  5. Update the posterior state estimation error covariance matrix.

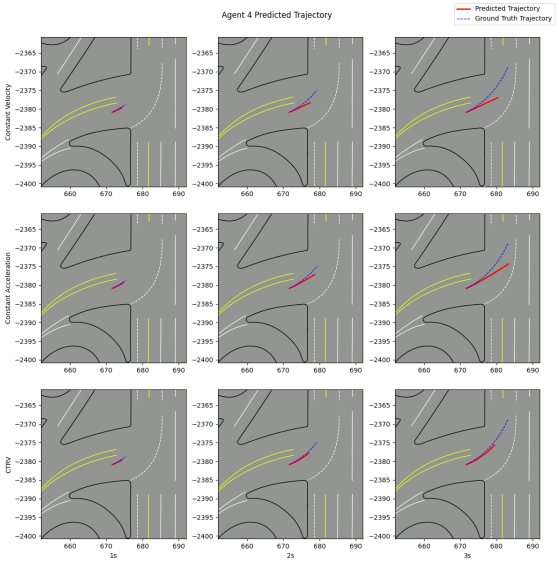
$$P(k) = (I - K(k)H)P'(k) \tag{10}$$

Where P denotes the State estimation error covariance matrix, Q represents the process noise covariance matrix, and R signifies the measurement noise covariance matrix.

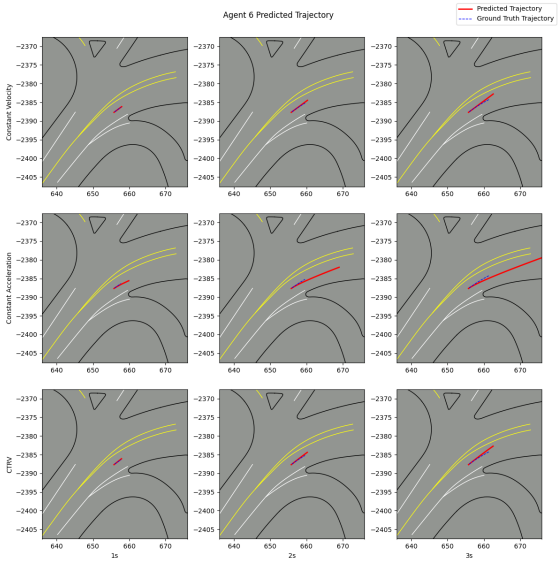
B More Experiment Results

Table 1: Numerical results of trajectory prediction.

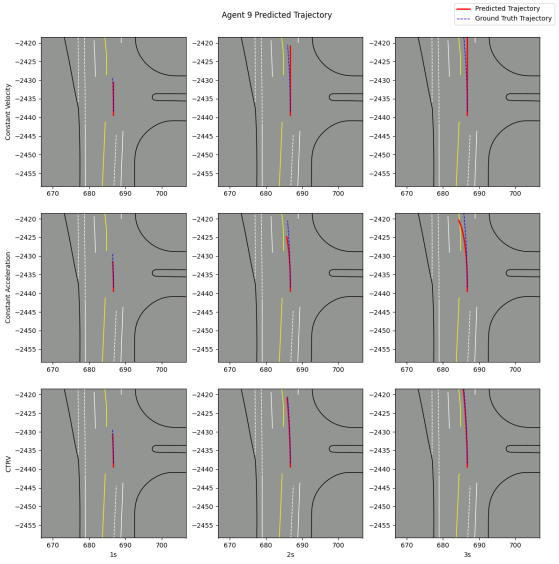
Horizon		1s						2s						3s					
Metric		ADE			FDE			ADE			ADE			ADE			ADE		
Model		Acc	Vel	CTRV	Acc	Vel	CTRV	Acc	Vel	CTRV	Acc	Vel	CTRV	Acc	Vel	CTRV	Acc	Vel	CTRV
Agents	**4**	0.66	0.80	0.79	1.03	1.42	1.40	1.20	1.76	1.70	2.50	4.08	3.81	2.13	3.32	3.08	5.51	8.64	7.72
	5	0.69	1.06	1.06	0.47	1.47	1.47	0.85	1.69	1.69	2.02	3.16	3.16	1.92	2.64	2.64	6.01	5.83	5.83
	6	0.78	0.13	0.14	2.26	0.25	0.28	3.41	0.38	0.40	9.93	1.02	1.04	7.77	0.75	0.77	22.61	1.84	1.86
	7	0.30	0.31	0.31	0.37	0.41	0.41	0.36	0.41	0.41	0.44	0.56	0.56	0.40	0.47	0.47	0.51	0.63	0.60
	9	1.47	1.16	1.16	2.11	1.23	1.22	2.37	1.11	1.02	4.32	0.87	0.39	3.51	1.26	1.01	7.06	2.66	2.23
	14	1.11	0.62	0.60	2.20	0.80	0.76	2.93	0.83	0.75	7.42	1.32	1.06	5.95	1.13	0.93	16.29	2.09	1.52
	15	1.87	0.91	0.90	3.88	1.17	1.13	5.33	1.39	1.25	13.78	2.68	2.13	10.86	2.44	1.98	29.29	6.48	4.85
	22	0.22	0.12	0.09	0.50	0.16	0.08	0.73	0.18	0.08	1.99	0.31	0.08	1.56	0.25	0.09	4.38	0.45	0.08



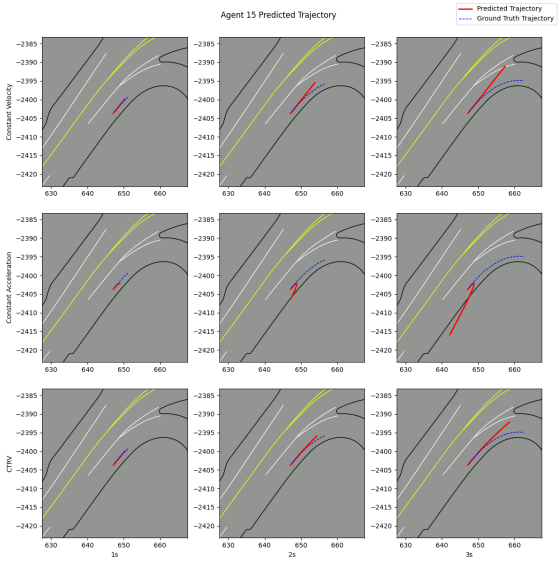
(a) Agent 4



(b) Agent 6



(c) Agent 9



(d) Agent 15

Figure 4: Visualization of the predicted trajectories.

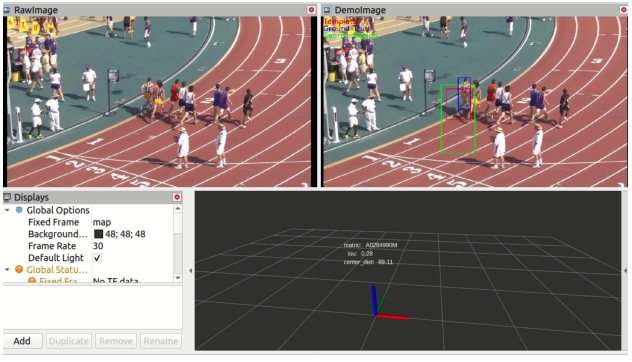


Figure 5: The running effect of Bonus task in Rviz.