Final Report:
Learning a Unified Policy for Locomotion with Eye-in-Hand Perception

Harry Freeman and Paul Nadan

I. Introduction

Mobile robotics lies at the intersection of three components: perception, manipulation, and locomotion. Much research has been done on the intersections of each pair of these components. Cameras attached to manipulators are used for eye-in-hand perception, which provides a close-up view that follows the object being manipulated. Cameras can also be attached to mobile robots to enable egocentric vision, allowing the robot to see and avoid obstacles as it moves. Finally, whole-body control is used to jointly carry out locomotion and manipulation on a single platform. In each case, coupling two different components can provide greater capabilities than using each in isolation. However, we believe that robots will ultimately need to rely on all three of these components at the same time in order to accomplish more complex tasks.

In this project, we hope to investigate the intersection of vision, locomotion, and manipulation. To do so, we consider a mobile robot with a camera mounted at the end of a mobile arm, tasked with tracking and approaching a visual target while avoiding obstacles. This task is challenging because the robot must keep the target in view and maintain a proximal distance to it, which requires simultaneously controlling the mobile arm to orient the camera and the legs to move towards the target. While a decoupled policy for each sub-task could likely achieve a reasonable level of performance, we hope to demonstrate that a unified policy for both locomotion and camera manipulation could lead to new useful behaviors and superior performance overall. We hope this will ultimately allow mobile robots to perform more complex tasks that require all three of locomotion, manipulation, and vision.

II. Prior Work

Mobile robots typically rely on a camera or another perceptive sensor that is fixed to the robot's body. With well-chosen camera placement, such a robot can have a clear view of the ground ahead, but a fixed placement can limit its awareness of its surroundings. For instance, [1] presents an end-to-end learned policy for a quadruped robot with a body-mounted camera. To avoid the computational costs of camera rendering during training, they instead provide their policy a height-map of the terrain. Once training has finished, they then apply a supervised learning approach to recondition the policy to use the raw camera image.

In contrast, robotic arms with cameras on the end effector (the "eye-in-hand" configuration) have been used for a wide range of tasks, such as surgery [2] and agriculture [3]. A technique called visual servoing [4] creates a feedback loop between the camera and manipulator to align or track a target in the camera view.

Finally, many researchers have integrated robotic arms with quadruped robots. In [5], such a robot is controlled using a whole-body hierarchical quadratic programming (HQP) approach during stationary manipulation tasks, but the authors resort to a decoupled strategy for the arm and legs during loco-manipulation. In contrast, [6] trains a joint policy to achieve both tasks using a technique called advantage mixing, where the manipulation and locomotion rewards are initially decoupled but converge as training progresses.

III. Approach

A. Mobile Robot

The mobile robot we constructed for our simulated experiments is a legged robot mounted with a mobile arm. The urdf we use for the legged robot is the 12 DoF ANYmal [7] quadruped, and the urdf for the arm is a 6 DoF wx250s from Interbotix Ros Manipulators [8]. The base of the arm is attached to the base of the legged robot with a fixed joint. The final result is a 5-limbed, 18 DoF mobile robot that we dub "The Pentapede" (Figure 1). A camera sensor is placed at the end of the robotic arm to allow the robot to visualize the environment.

Fig. 1. The Pentapede 18 DoF mobile robot.

B. Tracking Task

The objective is for The Pentapede to be able to track a moving target. For our target, we use a 0.1m-radius red sphere. The initial sphere location is randomized with x and y coordinates sampled uniformly from an 8m sided square

centered on the robot starting position and z coordinate sampled uniformly between 0.5m-2m. The sphere is also initialized with a target position that is sampled from the same distribution as the starting position, clipped at a max distance of 4m. The ball moves towards the target position at a speed of 0.5m/s. Once the ball reaches the target position, a new target position is randomly selected and the process repeats.

The task of the robot is two-fold: keep the sphere in view of the camera, and position the robot as close to the xy coordinates of the sphere as possible.

C. Obstacles

To make the navigation task more challenging, we add obstacles into the environment. The obstacles, meant to mimic trees in a forest, take the form of half-capsules with a fixed radii and heights of 0.2m and 1.5m respectively. A fixed number of 10 obstacles are randomly placed in the environment with a configured minimum distance of 1.0m separating them. An example of the tracking task in an obstacle-filled environment can been seen in Figure 2.

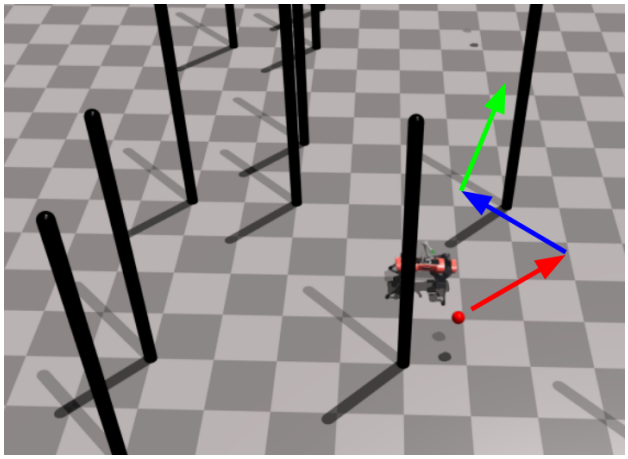

Fig. 2. The Pentapede navigating a "forest" of tall, tree-like obstacles. The robot must follow a ball that moves in a randomized trajectory, as shown by the arrows.

D. Observation Space

Although the simulation environment supports camera rendering, we found that this slows down the simulator significantly by a factor of 8. To work around this, we came up with an approach inspired by [1] that mimics detecting a ball in an image with the idea that it could later be replaced with image rendering by downstream fine-tuning or sim2real transfer. We first determine if the ball lies within the field of view of the camera. If it does, The Pentapede is told the direction and distance of the camera to the ball. If it does not, The Pentapede does not know where the ball is located. This assumption is justified because learning to detect the location of a red ball in a camera image could easily be done independently with segmentation or a CNN, and the distance could be determined using an RGB-D camera. Because object detection is not the focus of the projection, we used our simplified approach throughout training.

To determine if the ball lies in the camera field of view (FoV), we make a further simplifying assumption that the field of view is conical, such that the inequality in Equation 1 holds true for objects in view, where c_d is a unit vector along the camera axis, θ_{FoV} is the angular width of the camera field of view, and b_d is a unit vector pointing from the camera to the ball. We use a FoV of $80°$.

$$\alpha = \left[c_d \cdot b_d \geq \cos\left(\frac{\theta_{FoV}}{2}\right)\right] \tag{1}$$

We assume that obstacles are transparent and the agent can see the ball through the obstacles. We also assume that the agent knows where each obstacle in the environment is located relative to its pose. These assumptions were made for simplicity. The task and environment is already fairly complicated as the environment and paths of the ball are randomized each episode. Given more time, we would like to include occluding the sphere from view when behind obstacles and needing to detect obstacle positions with the camera.

Therefore, the observation space of our agent consists of α from Equation 1, the displacement of the ball with respect to the camera (if $\alpha = 1$, otherwise 0), the displacement of the ball with respect to the base of the agent (if $\alpha = 1$, otherwise 0), the displacement of all the trees with respect to the base of the agent, the previous actions taken, and the linear velocity, angular velocity, joint positions, and joint velocities of the agent, resulting in an observation space of size (100). As well, all of the observations are in the coordinate frame of the agent and not the world. This is to make it so that The Pentapede need not know its position in the environment in order to act properly.

In the future, we also would like to add temporal information into the observation space. We believe adding more previous actions and previous sphere positions will improve the robot's ability to search for the object.

E. Action Space and Rewards

The action space of our agent is the rotation angle of the joints. It is absolute rotation angle, not a delta. In the future, we plan on evaluating the advantages of using absolute vs delta. All actions are clipped to the range of [-1, 1]. The legged joints are then scaled to a rotation between $[-\frac{\pi}{2}, \frac{\pi}{2}]$, whereas the arm joints are scaled to

- waist: $[-\pi, \pi]$
- shoulder: $[-1, 1]$
- elbow: $[-1, 1]$
- forearm_roll: $[-\pi, \pi]$
- wrist_angle: $[-1, 1]$
- wrist_rotate: $[-1, 1]$

The reward function consists of a reward for both camera direction error and position error. Camera direction error is the error between the camera direction c_d and the direction between the camera and the ball b_d (Figure 3). Position error

is the error between the XY coordinates of the base of the robot r_p and the ball b_p. Our reward function is

$$r = \frac{\alpha}{1 + \|b_d - c_d\|^2} \cdot s_{dir} + \frac{\alpha}{1 + \|b_p - r_p\|^2} \cdot s_{pos} \quad (2)$$

where s_{dir}, and s_{pos} are tuned hyperparameters, which we set as 0.005 and 2.0 respectively. This is because we found the agent learned how to set the camera direction a lot more easily than navigating to the correct position, and we want to favor moving closer to the ball as that is the overall objective. α is from Equation 1 and represents if the ball is in the field of view of the camera; the reward is 0 if it cannot see the ball. The reward function is always greater than or equal to 0, which ensures that the robot will learn to stand in order to maximize episode length, since episodes terminate early if the robot falls.

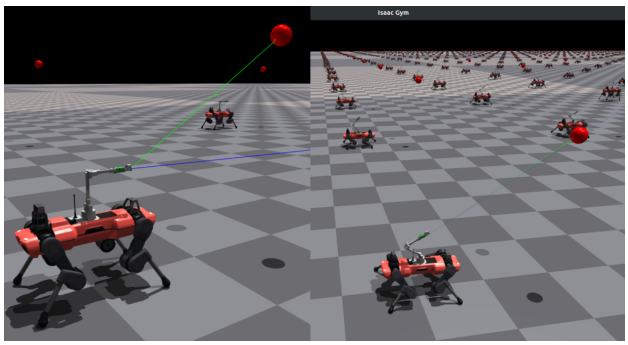

Fig. 3. Example camera direction error. Green line indicates the desired direction between camera and ball, blue line indicates direction camera is facing. Left image shows poor performance, right image shows strong performance.

F. Training

1) Training Phases: We divide our training process into two phases. In the first phase, the robot learns to locate and approach a stationary ball (with randomized position to prevent over-fitting). Once the policy is working well, we then fine-tune on the more difficult task of tracking and following a moving ball. Training to follow a moving ball leads to a noisy reward signal as the ball will at different times be moving towards or away from the robot, regardless of the robot's current actions. Thus initially training on a stationary ball helps the policy more quickly learn the basic skills of locomotion and camera tracking before the more challenging task is introduced.

2) Advantage Mixing: From [6], we experiment with advantage mixing to learn the optimal policy. We anticipate directly learning a unified policy for all joints will be a challenging task, as the overall goal of moving towards a target with the legs while keeping the target centered with the end-effector is quite complex. Advantage mixing will break down the problem by first incentivizing the robot to learn end-effector position and locomotion tasks independently, followed by the joining of the objectives into a single coupled task.

One difference between our implementation and the architecture in [6] is that for our baseline, we will not need to use regularized online adaptation. This is because Sim2Real is not in the scope of our project, and there is no need to train an adaptation module to learn environment encodings. Therefore, we will not be using an adaptation or encoder module and will instead directly feed states, actions, and obstacle locations directly into the policy network.

G. Simulation Environment

For simulation, we use Isaac Gym [9], NVIDIA's physics simulation environment for reinforcement learning. Isaac Gym is GPU optimized and allows for running tens of thousands of simulation environments in parallel, which we can take advantage of when training our simulated robot. For our reinforcement learning algorithm, we are using PPO as it was used in both [6] and [1] and is a common state-of-the-art off-policy algorithm. An example of the parallelized environments can be seen in Figure 4.

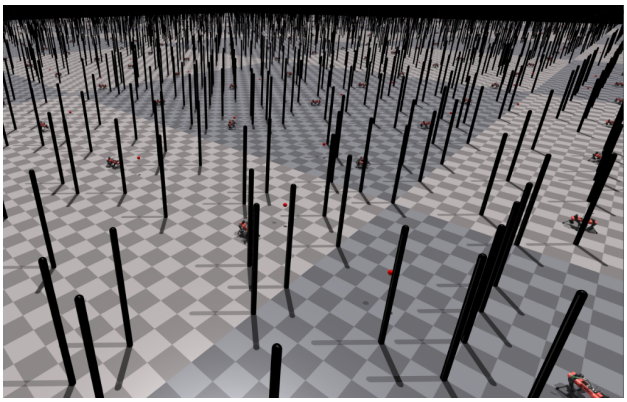

Fig. 4. Simultaneous training of hundreds of Pentapede robots in parallel using Isaac Gym.

IV. Experiments

After successful implementation of our approach from Section III, we conduct several trials to evaluate the benefits and shortcomings of our design decisions. First we evaluate the performance of our approach in an obstacle-free environment. This is to confirm that The Pentapede is able to accomplish the simpler task of tracking a ball without obstacles.

Then, we add the obstacles as described in Section III-C. With the obstacles present, we investigate the effect advantage mixing has on learning a policy with the mobile arm. We evaluate the robot's performance trained with and without advantage mixing to determine if it results in improved training time or if the robot is able to effectively learn a complex joint policy without it.

For both obstacle-free and obstacle-present experiments, we try to ascertain the value of placing a camera on a mobile arm on top of a legged robot, compared to a system with a fixed camera as in [1]. To do this, we modify the setup to have the camera at a fixed pose and orientation relative to

the robot by locking the arm joints, and evaluate how much worse the robot is at tracking the target, or if it is able to track it just as well.

For all experiments, we train the agent using PPO with 2048 simultaneous environments and a batch size of 2048. The remaining hyper-parameters are left at the defaults for the ANYmal example in [10].

Different experiments were trained for a different number of iterations which depending on when the the returned rewards stopped improving.

All videos of our qualitative results can be found at the playlist `https://www.youtube.com/watch?v=oKGWNEpsKVY&list=PL3E8TLYJg57LuGxonWckZubRBXV1HJqku`, as well as the folder `https://drive.google.com/drive/folders/1n6e6t_YuEqDD3CrWywXEMfGBqLL9GSDy`. Our source code is available at `https://github.com/hfreecmu/IsaacGymEnvs`.

A. Obstacle-Free Experiments

The returned rewards for the obstacle-free experiments can be seen in Figure 5 and Figure 6. Both quantitatively and qualitatively, the mobile arm outperforms the fixed arm, validating our approach in this simple environment.

Note: The reward function we used for these experiments is different than that presented in Eq. 2. It is the reward function used for our midterm report as described in Eq. 3, where α, β, s_{dir}, and s_{pos} are tuned hyperparameters, which we had as as 0.5, 0.1, 1, and 5 respectively. Because training takes a while, we ran out of time to test both with the updated reward function.

$$r = e^{-\frac{\|b_d - c_d\|^2}{\alpha}} \cdot s_{dir} + e^{-\frac{\|b_p - r_p\|^2}{\beta}} \cdot s_{pos} \tag{3}$$

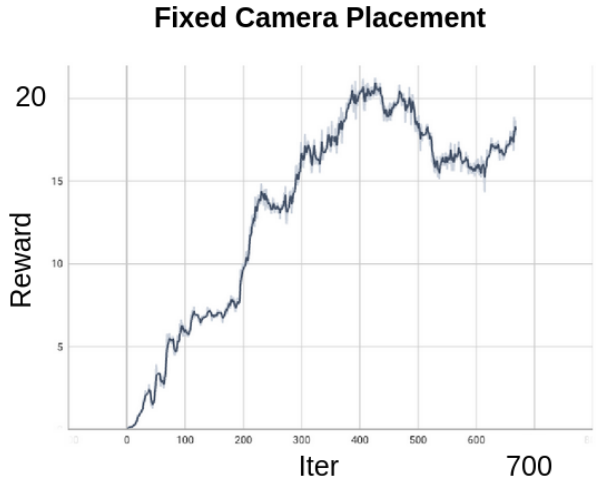

Fig. 5. CAPTION

As an additionally validity check for our design choices, we run tests fixing the ball height above the robot. While the experiment with the movable arm runs fine, the performance

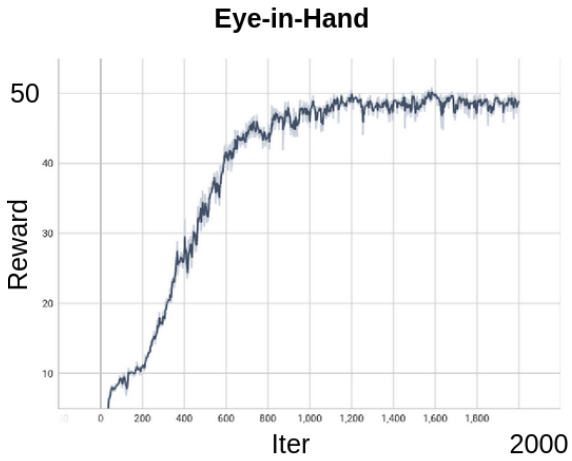

Fig. 6. CAPTION

of the fixed arm is significantly worse since the ball is above the robot and out of the line of sight of the camera. The robot learns to flip itself on its back to see the ball, and mobility is impaired (Figure 7).

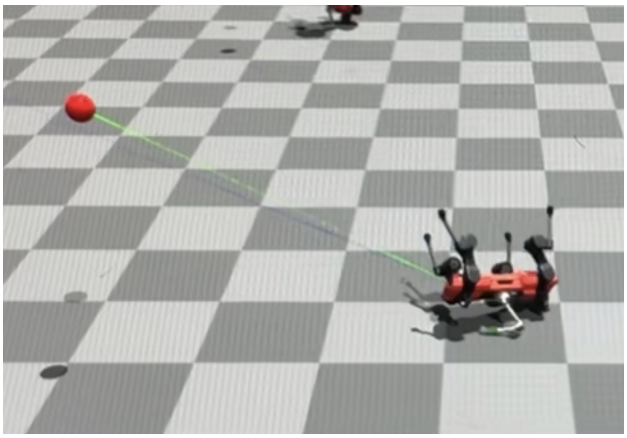

Fig. 7. When the camera is mounted on the robot body at a downward angle, the robot learns to flip itself over in order to continue tracking the ball.

B. Obstacle Experiments

To train the robot to follow the ball, we train a network on an initial stationary ball, once again using 2048 parallel environments and PPO. After 1500 epochs we add linear velocity to the ball. The robot successfully learns to follow the ball while keeping it in sight (Figure 8).

We compare three policies: eye-in-hand (Figure 9), eye-in-hand with advantage mixing (Figure 10), and fixed camera (Figure 11). For these trials, the eye-in-hand and fixed camera policies were not provided any reward for the camera direction, only for distance to the target. As shown in the training graphs, both eye-in-hand and fixed camera strategies reach a similar level of performance of around 200 reward on the moving ball task. In the supplemental videos, we see that the fixed arm policy learns to spin in a circle until the ball becomes visible, while the eye-in-hand policy searches

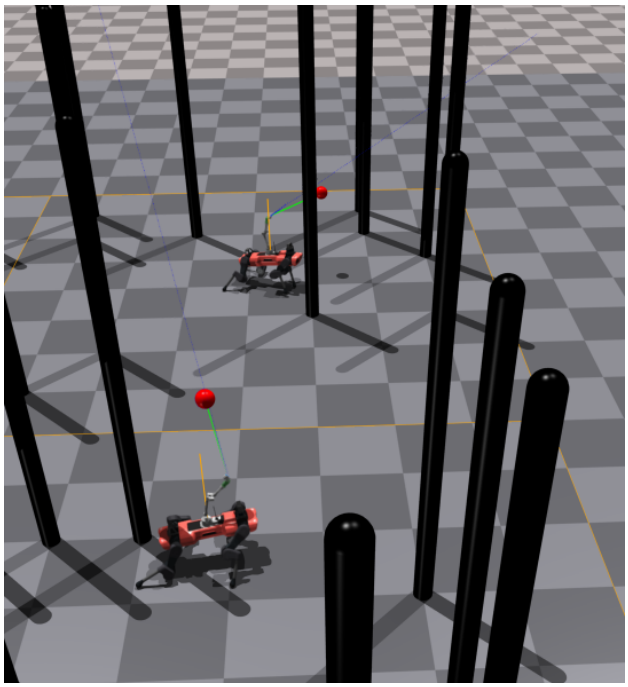

Fig. 8. Two simulated robots approaching their respective balls while maintaining camera tracking in a field of obstacles.

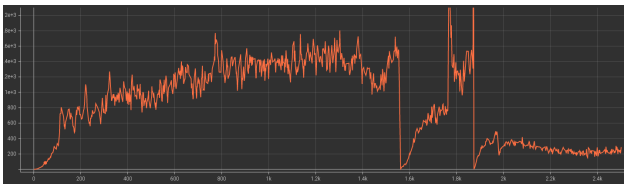

Fig. 9. Reward over the course of training for an eye-in-hand camera without advantage mixing. The first and second phase of training (after a crash) keep the ball stationary, while the third phase has a moving ball. The camera direction reward was not used for this trial.

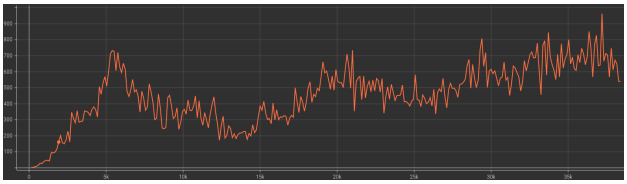

Fig. 10. Reward over the course of training for an eye-in-hand camera with advantage mixing. The first phase of training keeps the ball stationary, after which the trial was terminated due to lack of progress.

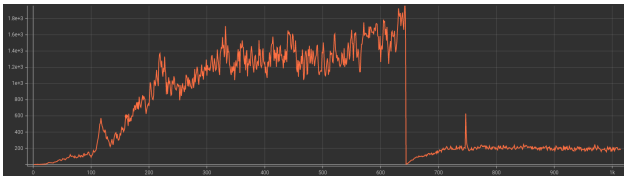

Fig. 11. Reward over the course of training for a fixed camera position relative to the robot. The first phase of training keeps the ball stationary, while the second phase has a moving ball. The camera direction reward was not used for this trial.

for the ball by rotating the arm, then aligns the body to the ball after it is located. The fixed camera policy also learns to execute more of a bounding motion while searching for the ball, while the eye-in-hand policy uses a more careful scuttle. This scuttling may be an effort to keep the arm stable, which is not as important during the searching phase.

One issue that both the eye-in-hand and fixed camera policies face is a failure to avoid collisions with obstacles. In both cases, the robot will frequently collide with an obstacle in an effort to move in a direct line towards the target. We are hopeful that this issue will eventually resolve itself with additional training time.

In contrast, the advantage mixing approach never learns to follow the ball, even when stationary. We attribute this to a poor breakdown of rewards between the arm and legs. Specifically, the legs are unaware of the ball direction until the arm learns to track the ball using the camera. Thus during the early stages of training when advantage mixing is in effect, the legs have no easy way to improve their reward.

To summarize, our initial hypotheses that advantage mixing would outperform eye-in-hand, and eye-in-hand would outperform fixed camera, were both invalidated. Instead, we saw worse performance for advantage mixing, and equal performance between the other two policies. We attribute this in part to an insufficiently challenging environment: without the presence of occlusions, the robot is able to solve the task effectively even with a fixed camera placement. We believe the addition of occlusions to the environment will better allow the eye-in-hand approach to differentiate itself.

V. Conclusion

In this project, we successfully trained a mobile robot to locate and follow a moving target. To do this, the policy learned several requisite skills, including searching for the ball, tracking the ball with the camera, and scuttling in the direction of the ball. However, although the robot is able to operate in the presence of obstacles, it still frequently collides with them instead of altering its route. Furthermore, while the use of a manipulator enabled more precise camera tracking, the robot with a fixed camera placement was still able to search for and follow the ball with a similar level of effectiveness, although it learned qualitatively different techniques for doing so. This suggests the need to increase the difficulty of the environment, so that the eye-in-hand approach can more easily distinguish itself.

To that end, one direction for future research is to add environmental occlusions to the simulation, such that the robot cannot see the ball when obstacles are in the way. This will prevent the arm alone from being able to locate the ball, and instead require the robot to use its legs to move around when searching for the ball. Another area for investigation is providing the policy with memory of its previous steps. This would allow the robot to rule out locations it has already searched when trying to find the ball, and also potentially improve its ability to navigate the

obstacle-filled environment. Finally, there is further work needed to transfer the approach from sim2real. This includes supervised learning to recondition the policy using the raw camera image as in [1], as well as using regularized online adaptation as in [6] to adapt to differences between the simulated and real environments.

REFERENCES

[1] A. Agarwal, A. Kumar, J. Malik, and D. Pathak, "Legged locomotion in challenging terrains using egocentric vision," in *6th Annual Conference on Robot Learning*, 2022.

[2] M. C. Capolei, H. Wu, N. A. Andersen, and O. Ravn, "Positioning the laparoscopic camera with industrial robot arm," in *3rd International Conference on Control, Automation and Robotics (ICCAR)*, pp. 138–143, 2017.

[3] T. Dewi, P. Risma, Y. Oktarina, and S. Muslimin, "Visual servoing design and control for agriculture robot; a review," in *International Conference on Electrical Engineering and Computer Science (ICECOS)*, pp. 57–62, 2018.

[4] D. Kragic, H. I. Christensen, *et al.*, "Survey on visual servoing for manipulation," *Computational Vision and Active Perception Laboratory, Fiskartorpsv*, vol. 15, p. 2002, 2002.

[5] G. Xin, F. Zeng, and K. Qin, "Loco-manipulation control for arm-mounted quadruped robots: Dynamic and kinematic strategies," *Machines*, vol. 10, no. 8, 2022.

[6] Z. Fu, X. Cheng, and D. Pathak, "Learning a unified policy for whole-body control of manipulation and locomotion," in *6th Annual Conference on Robot Learning*, 2022.

[7] "Meet ANYmal X, your ex-proof robotic inspector for the oil & gas and chemicals industries." `https://www.anybotics.com/anymal-autonomous-legged-robot/`.

[8] S. Wiznitzer, L. Schmitt, and M. Trossen, "interbotix_ros_manipulators." `https://github.com/Interbotix/interbotix_ros_manipulators`.

[9] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac Gym: High performance GPU-based physics simulation for robot learning," 2021.

[10] G. State and ynarang, "Reinforcement Learning Examples." `https://github.com/NVIDIA-Omniverse/IsaacGymEnvs/blob/main/docs/rl_examples.md`.