arXiv:1906.08236v1 [cs.RO] 19 Jun 2019

PyRobot: An Open-source Robotics Framework for Research and Benchmarking

Adithyavairavan Murali* Tao Chen* Kalyan Vasudev Alwala* Dhiraj Gandhi*
Lerrel Pinto Saurabh Gupta Abhinav Gupta

Facebook AI Research Carnegie Mellon University
https://www.pyrobot.org

Abstract

This paper introduces PyRobot, an open-source robotics framework for research and benchmarking. PyRobot is a light-weight, high-level interface on top of ROS that provides a consistent set of hardware independent mid-level APIs to control different robots. PyRobot abstracts away details about low-level controllers and inter-process communication, and allows non-robotics researchers (ML, CV researchers) to focus on building high-level AI applications. PyRobot aims to provide a research ecosystem with convenient access to robotics datasets, algorithm implementations and models that can be used to quickly create a state-of-the-art baseline. We believe PyRobot, when paired up with low-cost robot platforms such as LoCoBot, will reduce the entry barrier into robotics, and democratize robotics. PyRobot is open-source, and can be accessed via https://pyrobot.org.

1. Introduction

Over the last few years there have been significant advances in AI, specifically in the fields of machine learning, computer vision, natural language processing and speech. Most of these advancements have been fueled by high-capacity neural networks and the availability of large-scale datasets. However, an often overlooked reason for this fast-paced progress has been the development of a conducive research ecosystem. Platforms such as Caffe [35], PyTorch [52], TensorFlow [13] have reduced the entry barrier, which has democratized and accelerated research in these fields. For example, a new researcher in computer vision can get started with training state-of-the-art detectors using PyTorch and MSCOCO [41] in less than a day. Common platforms and datasets have also led to standardized evaluations and benchmarks which also helps quantify progress in these areas.

The field of data-driven robotics has also seen tremendous excitement and energy in the past several years [14, 15, 26, 31, 34, 39, 40, 43, 53–55, 65]. However, compared to other areas in AI, it has been relatively hard for a new researcher to get started and contribute to the progress in robotics. Why is that the case? One obvious reason is that researchers have to set up significant hardware infrastructure. This creates a high entry-barrier for researchers both in terms of financial cost and development time. Fortunately, there has been substantial progress on this front with the development of low-cost robots such as Blue [28], LoCoBot [31] and others [8, 64]. In fact, the cost of a robot is now comparable to that of the cost of a GPU! However even with these low-cost robots, getting started in robotics is still hard due to the lack of research platforms and a self-sustaining ecosystem.

Frameworks such as ROS [56] have made setting up robots substantially easier by providing a common mid-level communication layer and tools that are agnostic to low-level hardware and program context. However, there are two issues with such open-source frameworks:

ROS requires expertise: Dominant robotic software packages like ROS and MoveIt! are complex and require a substantial breadth of knowledge to understand the full stack of planners, kinematics libraries and low-level controllers. On the other hand, most new users do not have the necessary expertise or time to acquire a thorough understanding of the software stack. A light weight, high-level interface would ease the learning curve for AI practitioners, students and hobbyists interested in getting started in robotics.

Lack of hardware-independent APIs: Writing hardware-independant software is extremely challenging. In the ROS ecosystem, this was partly handled by encapsulating hardware-specific details in the Universal Robot Description Format (URDF) which other downstream services

*The first four authors contributed equally to this paper.

could read from. Yet, from the perspective of high-level AI applications, most robotics code is still hardware dependent. As a community, we lack a research platform and a common API that we can use to share code, datasets and models.

In this white-paper, we attempt to tackle these challenges via an open-source research platform – **PyRobot**. PyRobot is a light weight, high-level interface on top of ROS that provides hardware independent mid-level APIs and high-level examples for manipulation and navigation. PyRobot also provides libraries for hand-eye calibration, tele-operation, trajectory tracking, and SLAM-based navigation. We believe PyRobot combined with the recently released LoCoBot robot will reduce both the financial cost and development time – leading to democratization of data-driven robotics. The hardware-independent API will lead to development of code and datasets that can be shared across the community. While the current PyRobot release interfaces with LoCoBot and Sawyer, we plan to release integration with several new robots like the UR5 [2] and Franka [5], and simulator platforms like MuJoCo [60] and Habitat [47].

2. PyRobot Framework

PyRobot is a python-based robotics framework that isolates the ROS system [56] from the user-end and supports the same API across different robots (see Figure 1 for an overview). Essentially, it provides a python wrapper around the mid-level features provided by ROS and the low-level C++/C controllers and driver backends. PyRobot has common utility functions for all robots, such as joint position control, joint velocity control, joint torque control, cartesian path planning, forward kinematics and inverse kinematics (based on the robot URDF file), path planning, visual SLAM, among other features. Though it abstracts away the complexity of the underlying software stack, users still have the flexibility to use components at varying levels of the hierarchy, such as commanding low-level velocities and torques by-passing a planner. We summarize the design philosophy behind PyRobot below.

Beginner-friendly. Ideally, new users should be able to start commanding a robot in just a few lines of code, as shown in the Listing 1, without learning ROS or the underlying software and firmware stack.

Hardware-agnostic design. PyRobot is designed to easily accommodate common robotic manipulators and mobile bases. Currently, it supports LoCoBot, a low-cost mobile robot with a 5-DOF manipulator and a Sawyer robot. Each robot has a YACS [10] configuration file that specifies the necessary robot-specific parameters: joint names, ROS topics to get state and set commands, base frame, end-effector frame, planner configuration, inverse kinematics solution tolerance, whether it has an arm or base or camera, *etc*. A PyRobot object requires the config file for initialization. As shown in Listing 1, the Sawyer robot can be commanded in a manner identical to that of LoCoBot.

```python
# LoCoBot - Arm
from pyrobot import Robot
bot = Robot('locobot')
target_joints = [0, 0, 0, 0, 0]
bot.arm.set_joint_positions(target_joints)

# LoCoBot - Base
target_position = [1, 1, 1]
bot.base.go_to_absolute(target_position)

# Sawyer
from pyrobot import Robot
bot = Robot('sawyer',
            use_arm=True,
            use_base=False,
            use_camera=False,
            use_gripper=True)
target_joints = [0, 0, 0, 0, 0, 0, 0]
bot.arm.set_joint_positions(target_joints)
```

Listing 1: PyRobot example for position control on LoCoBot and Sawyer.

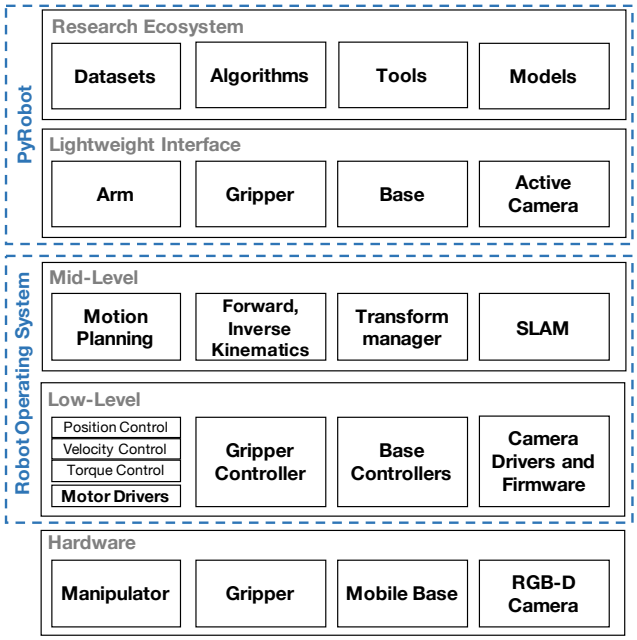

Figure 1: Overview of PyRobot system architecture.

Open Source. Robotics systems development has typically been constrained to robotics experts in academia and industry with access to expensive and niche robotics systems. However, the extensive scope of artificial intelligence

requires strong collaboration between researchers to build and maintain these large systems and one can contribute to all layers of the stack with open sourcing. Apart from the open software, LoCoBot works as an affordable open hardware that can be easily assembled for use with PyRobot. While simulation is useful for software testing and running experiments, writing software that works on the real robot is the eventual goal of the field and has severe challenges. As more developers have access to both open hardware and software, high quality applications tested on real robots can be publicly shared.

3. Supported Hardware and Simulators

PyRobot is currently integrated with the following robots. In addition to real robots, PyRobot can also be used to control robots in simulators like Gazebo.

LoCoBot: LoCoBot, shown in Figure 2 (left), is a low-cost mobile manipulator platform built for easy setup and benchmarking robot learning research. It consists of a Trossen Widow X robotic arm [9] assembled with Dynamixel XM-430 and XL-430s servo motors. The arm has five degrees of freedom (DOFs) - with a working payload of $0.2\,\mathrm{kg}$ and a maximum reach of $0.55\,\mathrm{m}$. The robot comes in two versions, with the arm rigidly mounted on a Kobuki mobile base [7]. The Kobuki base is about $0.12\,\mathrm{m}$ high with payload capacity of around $4.5\,\mathrm{kg}$. For visual perception, an Intel Realsense D435 RGBD camera [6] is mounted with a pan-tilt attachment at a height of about $0.6\,\mathrm{m}$ above the ground. An automatic camera calibration routine is implemented in the software suite. LoCoBot also comes with a Intel NUC (i5, 8GB RAM) machine rigidly attached on the base, which could be used for on-board compute. Kobuki base is powered through its own battery that can run base for about 2 hours. We use a 185 Wh battery pack [3] to power the arm, pan-tilt mount, and the on-board computer. On a full charge, the complete system is able to run for 50-60 minutes. LoCoBot-Lite, shown in Figure 2 (right), is a cheaper version of LoCoBot that uses the Create2 base [4] instead of the Kobuki base.

Sawyer: The Sawyer is a 7-DOF collaborative robot arm from Rethink Robotics [1]. PyRobot interfaces with the Intera SDK provided with the Sawyer.

Simulators: PyRobot currently supports Gazebo simulator [37], a 3D rigid body simulator popular in the robotics community. For LoCoBot and LoCoBot-Lite, PyRobot supports tight integration with Gazebo *i.e.*, the same code can be run on both Gazebo and the real robot.

4. PyRobot Controllers

While a number of robots come with their own implementations for low-level control, PyRobot implements basic controllers for differential drive bases. It also interfaces with planners such as MoveIt! [20] and Movebase [49]. We measure the performance of these controllers and planners implemented in PyRobot for the LoCoBot base and arm.

Figure 2: LoCoBot (left) and LoCoBot-Lite (right). Both robots have a 5 DOF arm mounted on top of a mobile base (Kobuki or Create2). Robots are equipped with a RGB-D camera mounted on a pan-tilt stand. Robots come with a battery pack and an on-board computer.

4.1. Accuracy of Base Control

PyRobot implements position controllers to command the robot base to a desired target position (parameterized as a 3-DOF pose, (x, y) location of the base and its heading θ: $[x, y, \theta]$). We implement the following three controllers:
DWA Controller from Movebase: We implemented Dynamic Window Approach Controller (DWA) [27] for our robot through Movebase [49] navigation engine. In this approach, we repeatedly sample a discrete sequence in the robot's control space with the highest score and execute the sequence until the target is reached.
Proportional Controller: We decompose the motion into an on-spot rotation, linear motion and a final on-spot rotation at the target location. Each segment of this motion is

Table 1: Base position control performance for LoCoBot and LoCoBot-Lite. We report translation and rotation error for different motion types for the different controllers for base position control implemented in PyRobot. Lower errors are better.

	Error with respect to motion capture			Error with respect to odometry		
Controllers	ILQR	Proportional	Movebase	ILQR	Proportional	Movebase
LoCoBot						
Linear motion						
Translation (mm)	17 ± 5	46 ± 23	89 ± 16	3 ± 1	41 ± 32	102 ± 2
Rotation (deg)	0.43 ± 0.25	1.77 ± 1.46	10.81 ± 2.19	0.12 ± 0.10	1.65 ± 1.37	10.63 ± 2.19
Rotation motion						
Translation (mm)	6 ± 0	6 ± 4	4 ± 2	0 ± 0	5 ± 1	2 ± 1
Rotation (deg)	1.32 ± 0.68	2.48 ± 0.98	12.53 ± 1.09	1.45 ± 0.24	2.54 ± 1.02	13.08 ± 1.18
Combined motion						
Translation (mm)	16 ± 2	65 ± 52	78 ± 2	6 ± 1	55 ± 50	87 ± 15
Rotation (deg)	0.29 ± 0.19	3.2 ± 2.69	11.59 ± 1.3	0.84 ± 0.20	2.35 ± 2.94	11.65 ± 1.63
LoCoBot-Lite						
Linear motion						
Translation (mm)	144 ± 8	142 ± 7	260 ± 81	9 ± 5	34 ± 5	99 ± 31
Rotation (deg)	1.79 ± 1.59	2.82 ± 0.52	7.34 ± 8.19	1.6 ± 1.5	1.61 ± 0.34	5.21 ± 3.13
Rotation motion						
Translation (mm)	3 ± 2	3 ± 2	3 ± 1	2 ± 2	3 ± 3	3 ± 1
Rotation (deg)	6.97 ± 1.71	3.07 ± 3.47	9.94 ± 1.46	1.44 ± 1.12	4.59 ± 2.78	3.42 ± 1.66
Combined motion						
Translation (mm)	123 ± 7	99 ± 4	230 ± 57	5 ± 6	93 ± 19	93 ± 21
Rotation (deg)	2.8 ± 1.68	1.19 ± 0.95	5.87 ± 8.22	2.57 ± 1.31	1.57 ± 1.15	4.18 ± 3.45

Figure 3: LoCoBot is low-cost and hence scalable.

executed using a proportional controller that applies velocities proportional to the tracking error. For smooth motion, we bound the velocities and the change in velocities.

Linear Quadratic Regulator: We analytically compute a trajectory (a sharp one that breaks the motion into on-spot rotation, straight motion and a final on-spot rotation; or a smooth one by fitting a bézier curve between the stating state and the ending state). We sample this trajectory to obtain a state trajectory using constraints on maximum linear and angular velocities. We linearize the dynamics of the robot (assumed to be a bicycle model [16]) around this state trajectory, and construct a LQR feedback controller [16] to track this state trajectory.

We conducted trials on the robot to quantify the accuracy of each of these different position controllers on both LoCoBot and LoCoBot-Lite. We measured accuracy using the difference in commanded state *vs*. the achieved state as measured using a Vicon motion capture system. The error was factored into translation (difference in (x, y) location), and rotation (difference in the heading θ). We report these errors in Table 1. We group trials into the following three categories: *a) Linear motion*: 5 trials each with targets $2\,\mathrm{m}$ in front ($[2, 0, 0]$), or $2\,\mathrm{m}$ behind ($[-2, 0, 0]$); *b) On-spot rotation*: 5 trials each with target being left rotation by $\pi/2$ ($[0, 0, \pi/2]$), right rotation by $\pi/2$ ($[0, 0, -\pi/2]$); *c) Combined linear and rotation motion*: 5 trials each with targets $[1, 1, 0]$ and $[-1, -1, 0]$.

Table 1 reports translation and rotation errors for the different controllers for the two robots for these different cases. We generally note that errors are lower for LoCoBot *vs*. LoCoBot-Lite. Additionally, LQR and proportional controller generally perform better than the DWA controller from Movebase. As all these controllers close the loop on

Table 2: Locobot Arm Pose Repeatability

Std Dev.(mm)	Poses				
	1	2	3	4	Home
x	0.12	0.13	0.07	0.11	0.15
y	0.13	0.07	0.10	0.14	0.27
z	0.21	0.33	0.22	0.31	0.24
Repeatability (mm)	0.41	0.58	0.33	0.50	0.52

the base odometry, we additionally include errors with respect to base odometry in right part of the table. We observe that the LQR controller is more effective at closing the loop.

PyRobot also implements trajectory tracking (using feedback controllers as described above). We show qualitative comparisons between different controllers in Figure 4.

4.2. Repeatability Tests for Manipulator

Compared to expensive industrial and collaborative robots, low-cost manipulators like LoCoBot suffer from control errors that can be attributed to a range of factors: manufacturing and assembling error, gear backlash, hardware execution error, kinematics inaccuracy, hand-eye calibration error, motor wear and tear, etc. The position-control repeatability was analyzed by commanding the arm to 4 different 3D poses (and the home pose) in a 2D grid at a fixed height without carrying a payload for a total of 10 repetitions per pose. The ground truth positions were measured using a Vicon motion capture system at $120\,\mathrm{Hz}$. The arm always started at the home pose (when the joint angles are all 0) before moving to the commanded end pose. The results are summarized in Table 2. Overall, the arm had a repeatability error of $0.33\,\mathrm{mm}$ to $0.58\,\mathrm{mm}$, computed based on ISO9283 standard. Poses 1 and 3 were closer to the robot torso and had lower error compared to Pose 2 and 4 where the arms were extended at the extremities of the workspace. The standard deviation along the z axis was also higher across all poses due to gravity. For comparison, the Sawyer and UR5 robots are reported to have a repeatability of $0.1\,\mathrm{mm}$ [1, 2]. The position control in the initial release only relies on proprioceptive feedback, and using feedforward model-based control in future release could reduce the error further. The PID gain settings are exposed to the user for more specialized robot or task-specific tuning.

5. High-Level AI Applications

We discuss implementation of a few example high-level AI applications through the PyRobot API.

5.1. Visual SLAM

Visual SLAM algorithms provide more accurate odometry as compared to odometry that is derived purely from inertial sensors on the base. We deployed ORB-SLAM2 [51],

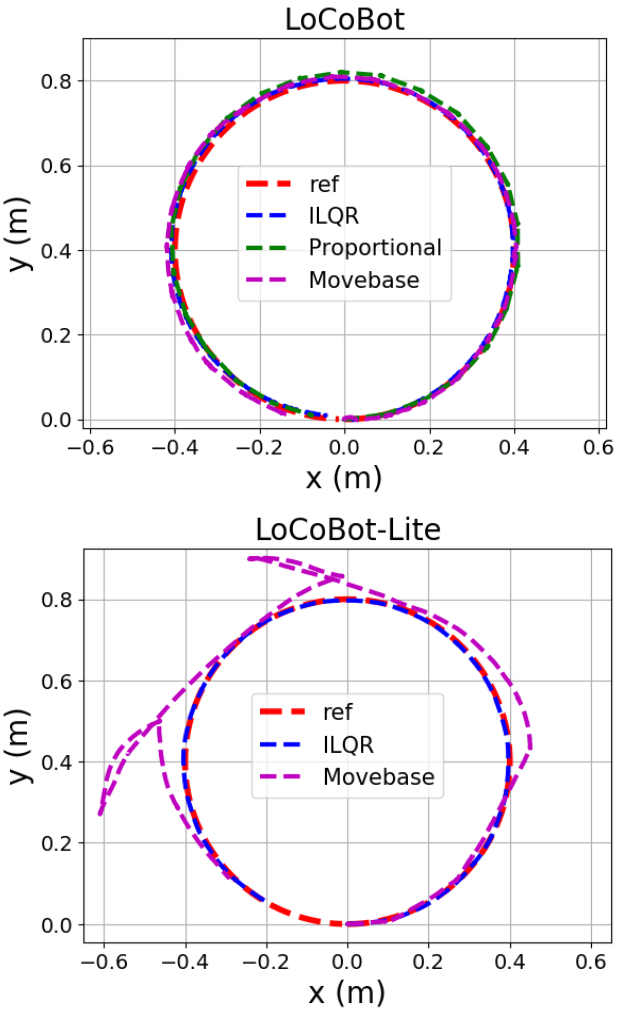

Figure 4: Qualitative comparisons for trajectory tacking for LoCoBot and LoCoBot-Lite. Reference trajectory (a circle of radius $0.4\,\mathrm{m}$) is shown in red.

a leading visual SLAM systems in the PyRobot library. ORB-SLAM2 is a feature-based indirect visual SLAM system that uses ORB features to perform tracking, mapping, and loop closing. We adapt the open-source ORB-SLAM2 code into a ROS package. This package saves RGB and depth images of the keyframes and continuously publishes camera trajectory and camera pose. PyRobot uses this published pose information to return the robot base state and trajectory. This state derived from visual SLAM can be used in downstream controllers or algorithms for more accurate behavior. PyRobot also supports dense map reconstruction, by integrating depth image observations using the ORB-SLAM2 estimated camera pose. This can be used for motion planning for navigation tasks.

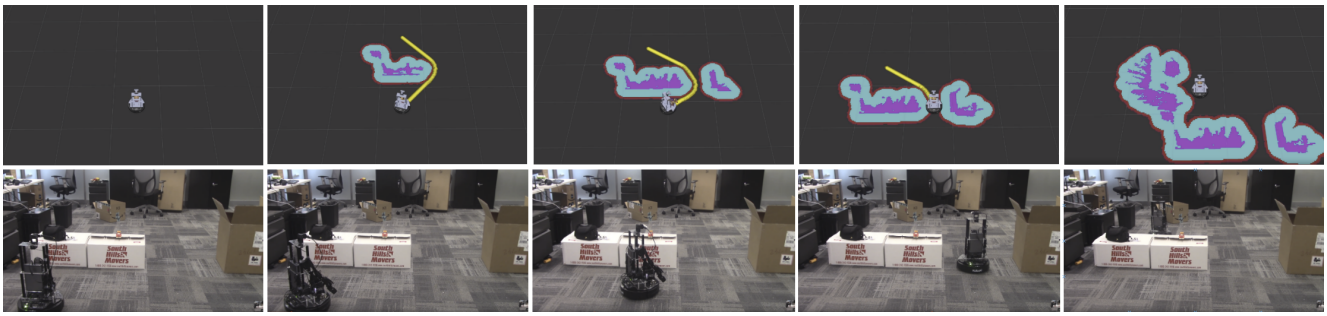

Figure 5: An example of Navigation via SLAM and Path Planning. First row corresponds to the 2-D map constructed using the on-board SLAM and the second row corresponds to the actual motion of the robot.

5.2. Navigation via SLAM and Path Planning

We deployed Movebase [49] ROS package on LoCoBot and LoCoBot-Lite for safe navigation in environments with obstacles. We use the occupancy map as obtained from visual SLAM, to compute a 2D cost-map that denotes regions of the environment where the robot is safe to move. Movebase uses this cost-map to generate collision free trajectories to goals specified in the environment. These trajectories can be executed using any of the controllers implemented in PyRobot. These steps are run continuously, and the plan is updated if it becomes infeasible as the robot perceives previously unseen parts of the environment.

5.3. Learned Visual Navigation

We deploy learned policies for visual navigation on LoCoBot using PyRobot API. We work with the cognitive mapping and planning policy (CMP) from Gupta *et al*. [32]. Given an input goal location, CMP policy takes in the current image from the on-board camera to output one of four macro-actions (stop, turn left, turn right or go straight). We use the base position control interface in PyRobot API to execute these actions. Listing 2 shows simplified code, and Figure 6 shows frames from a sample execution.

5.4. Grasping

We deploy a learned-based grasping algorithm to grasp objects placed on the ground from RGB images using the PyRobot API. The model is trained on data from people's homes [31] and is robust to a wide variety of objects and backgrounds. This model outputs a grasp in the image space. This grasp is parameterized by 2D location in the image and the gripper orientation. We convert this 2D location and orientation into the *grasp position* (3D location and orientation) using known camera parameters, and the depth image. We command the robot to the *pre-grasp location*, that is a few centimeter above the grasp position, lower the arm to reach the object, and close the gripper to grasp the object. Listing 3 shows simplified code, and Figure 7 shows sample grasps using the LoCoBot.

5.5. Pushing

We deploy a heuristic-based pushing algorithm using PyRobot. It relies on the depth sensor, and thus the quality of the pushing depends on how well the stereo-based depth sensor behaves in different background. To achieve the best performance, it is best to place the robot on a floor with non-uniform texture.

The algorithm can be summarized with the following steps: (1) Move the arm out of the camera's field of view. (2) Filter the point cloud seen by the RGBD camera, specifically removing points too far away and those that correspond to the floor by coordinate thresholding. (3) Project the remaining point cloud onto the xy-plane and use DBSCAN [24] algorithm to automatically cluster the projected points. (4) Randomly select one cluster and choose a random push-start point on the enclosing bounding box of the cluster. (5) Move the gripper to the push-start point and move the gripper horizontally towards the center of the cluster. Listing 4 shows simplified code.

6. Related Work

Robotics Software Design. The robotics community has embraced a layered hierarchical software design from the early days [17] and re-usability has been a core design principle [45]. We refer readers to Tsardoulias and Mitkas [61] for a comprehensive review. There have been several motion planning libraries such as OpenRave [23], MoveIt! [20], OMPL [59] which provide hardware-agnostic core functionalities that can be compiled for each specific robot. In the likes of ROS, there have also been robotics ecosystems, such as OROCOS [18] and the Microsoft Robotics Studio that support kinematic libraries, distributed processes, state machines for the real time control of robots.

Low-cost Mobile Manipulators. There has been very limited research on learning on low-cost robots, given that most researchers use standard industrial or collaborative

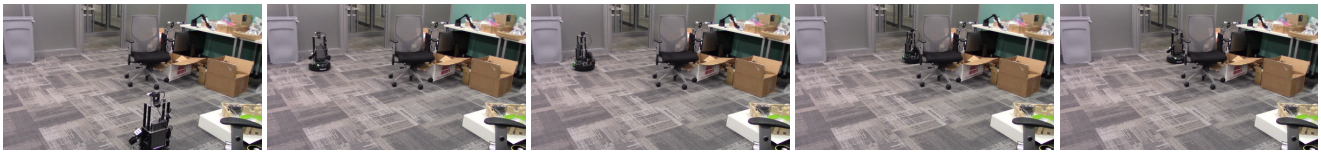

Figure 6: Snapshots from a run of visual navigation policy (CMP [32]) deployed on LoCoBot. See project website for videos.

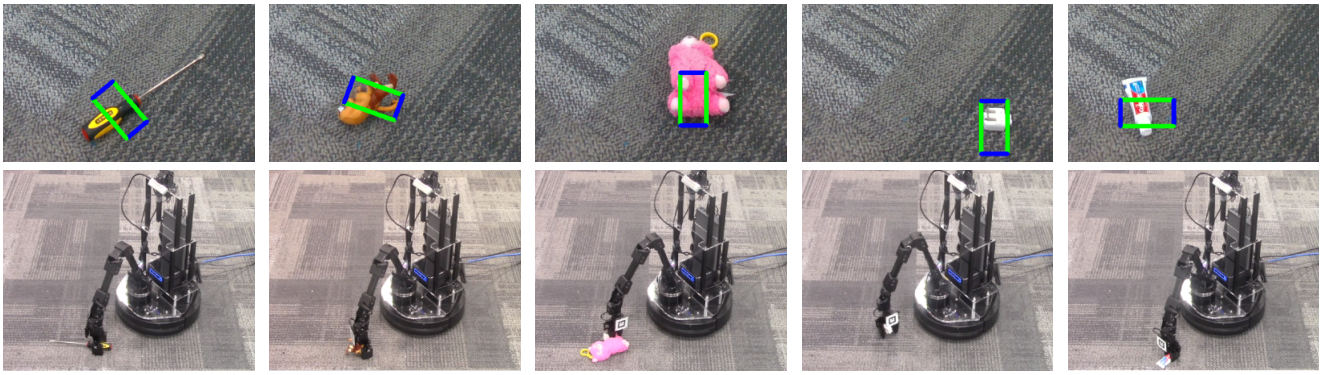

Figure 7: Grasps selected by the grasp model and execution by the robot.

robots. Deisenroth *et al*. [22] used model-based RL to teach a cheap inaccurate 6 DOF robot to stack multiple blocks and a previous iteration of LoCoBot was used in Gupta *et al*. [31] to learn visual grasping policies with real data collected in people's homes. Recently, Gealy *et al*. [28] proposed a compliant low-cost arm using quasi-direct drive actuation.

Open Source Manipulators. There has been very limited work in open sourced manipulators. Raven is a open architecture surgical research robot [57]. Recently, the Open Manipulator project from Robotis allows one to build their own low cost robot with custom kinematics and design [8].

Research Ecosystems in AI Fields. Research in a number of AI fields has benefited from there being common tasks (such as object detection in computer vision or parsing in NLP), common datasets (such as BSDS [50], ImageNet [58], PASCAL VOC [25] and MSCOCO [41] in computer vision, or Penn Tree Bank [48], GLUE [62], SentEval [21] and WMT in NLP, *etc*.), and common code bases to experiment with (DPMs [29], Caffe [35], Stanford CoreNLP [46], spaCy [33], *etc*.). While some people argue that such use of common tasks and datasets can prevent creative progress, at the same time, it has lead to rapid progress in these fields, as researchers can quickly replicate results and build upon each other work.

Benchmarking in Robotics. Benchmarking in robotics is extremely challenging given the vast scope of applications and diversity of physical test conditions (hardware, objects, environment, etc.). It is a well acknowledged concern within the robotics community that we are yet to develop reliable benchmarking metrics that can be widely adopted to quantify research progress. Several workshops have tried to stimulate discourse towards this end [11, 12] and different task specific metrics have been proposed for grasping [42], gripper design [38], SLAM [12], etc. Research has also benefited from creating object datasets with shape and grasp information, such as the Columbia Grasp Database [30], DexNet [44] and KIT Object Models [36], which could be used for perception and motion planning. The YCB dataset went a step further by distributing a physical dataset of household and kitchen objects with corresponding meta data (shape, RGBD scans, etc) [19]. While there is no consensus yet on benchmarking in robotics, we hope that the combination of PyRobot and LoCoBot will facilitate further discussion.

7. Discussion

In this paper, we describe the PyRobot framework, which provides a high-level hardware independent API to control different robots. We believe PyRobot when combined with low-cost robots such as LoCoBot, will reduce the barrier to entry into robotics. In the immediate future, we will continue to grow the functionality in PyRobot such as by interfacing with simulators (like AI Habitat [47], Gibson [63] and MuJoCo [60]), improving controllers such as be implementing gravity compensation for LoCoBot. But more broadly, we believe PyRobot will lead to the develop-

ment of a research and teaching ecosystem.

PyRobot for robotics instruction. Having a beginner-friendly and open architecture is great for robotics education, as affordable robotic setups with LoCoBot and PyRobot could easily be assembled and scaled for hands-on instruction. 10 LoCoBots were used in the Spring 2019 offering of 16-662: Robot Autonomy (by Professor Oliver Kroemer) in the Robotics Institute at CMU, to support homework assignments and projects. We believe many more such courses will follow.

PyRobot as a research ecosystem. Compared to other fields, benchmarking in robotics is challenging due to several reasons. PyRobot's unified API and LoCoBot's standard hardware, will allow researchers to share their high level algorithmic implementations, models and datasets collected on a real robot. This will allow researchers to collaborate and iterate faster on robotics applications. We will continue to expand the set of pre-trained models. Hopefully, other researchers will find the PyRobot framework useful and contribute their models for others to use as well.

8. Acknowledgements

We would like to thank Soumith Chintala for countless discussions and providing software engineering guidance. We would also like to thank Deepak Pathak and Shubham Tulsiani for testing, advice and discussions. Finally, we would like to thank Oliver Kroemer, Timothy Lee and Mohit Sharma for introducing LoCoBots in teaching *16-662: Robot Autonomy* at CMU, and Justin MacEy for helping with the motion capture experiments.

References

[1] Sawyer technical specifications. https://www.rethinkrobotics.com/sawyer/tech-specs/, 2016. 3, 5

[2] Ur5 technical specifications. https://www.universal-robots.com/media/50588/ur5_en.pdf, 2016. 2, 5

[3] Battery pack. https://www.maxoak.net/laptop-power-bank/show/11.html, 2018. 3

[4] Create2 mobile base. https://www.irobot.com/about-irobot/stem/create-2, 2018. 3

[5] Franka emika specification. https://www.franka.de/, 2018. 2

[6] Intel realsense d435 rgbd camera. https://click.intel.com/intelr-realsensetm-depth-camera-d435.html, 2018. 3

[7] Kobuki mobile base. http://kobuki.yujinrobot.com/about2/, 2018. 3

[8] Open manipulator project. http://emanual.robotis.com/docs/en/platform/openmanipulator_x/overview/, 2018. 1, 7

[9] Trossen widow x robotic arm. https://www.trossenrobotics.com/widowx-200-robot-arm-mobile-base.aspx, 2018. 3

[10] YACS – yet another configuration system. https://github.com/rbgirshick/yacs, 2018. 2

[11] ICRA workshop on benchmarks for robotic manipulation. http://www.ycbbenchmarks.com/ICRA2019_workshop, 2019. 7

[12] ICRA workshop on dataset generation and benchmarking of slam algorithms for robotics and vr/ar. https://sites.google.com/view/icra-2019-workshop/home, 2019. 7

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 1

[14] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016. 1

[15] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, J. T. Szymon Sidor, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018. 1

[16] K. J. Åström. *Introduction to stochastic control theory*. Courier Corporation, 2012. 4

[17] R. Brooks. A robust layered control system for a mobile robot. *AI Memo 864*, 1985. 6

[18] H. Bruyninckx. Open robot control software: the orocos project. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2001. 6

[19] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics & Automation Magazine*, 1070(9932/15):36, 2015. 7

[20] S. Chitta, I. Sucan, and S. Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012. 3, 6

[21] A. Conneau and D. Kiela. Senteval: An evaluation toolkit for universal sentence representations. *arXiv preprint arXiv:1803.05449*, 2018. 7

[22] M. P. Deisenroth, C. E. Rasmussen, and D. Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. *RSS*, 2011. 6

[23] R. Diankov and J. Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 2008. 6

[24] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. Density-based spatial clustering of applications with noise. In *Int. Conf. Knowledge Discovery and Data Mining*, volume 240, 1996. 6

[25] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, Jan. 2015. 7

[26] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017. 1

[27] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997. 3

[28] D. Gealy, S. McKinley, B. Yi, P. Wu, P. Downey, G. Balke, A. Zhao, M. Guo, R. Thomasson, A. Sinclair, P. Cuellar, Z. McCarthy, and P. Abbeel. Quasi-direct drive for low-cost compliant robotic manipulation. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2019. 1, 7

[29] R. B. Girshick, P. F. Felzenszwalb, and D. McAllester. Discriminatively trained deformable part models, release 5. http://people.cs.uchicago.edu/ rbg/latent-release5/. 7

[30] C. Goldfeder, M. Ciocarlie, H. Dang, and P. Allen. The columbia grasp database. In *International Conference on Robotics and Automation (ICRA)*, pages 1710–1716. IEEE, 2009. 7

[31] A. Gupta, A. Murali, D. Gandhi, and L. Pinto. Robot learning in homes: Improving generalization and reducing dataset bias. In *Advances in neural information processing systems*, 2018. 1, 6, 7

[32] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 6, 7

[33] M. Honnibal and I. Montani. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 2017. 7

[34] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. In *Science Robotics*, volume 4, 2019. 1

[35] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 1, 7

[36] A. Kasper, Z. Xue, and R. Dillman. The kit object models database: An object model database for object recognition, localization and manipulation in service robotics. *IJRR*, 2012. 7

[37] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004. 3

[38] G. Kragten, A. Kool, and J. Herder. Ability to hold grasped objects by underactuated hands: Performance prediction and experiments. In *International Conference on Robotics and Automation (ICRA)*, pages 2493– 2498. IEEE, 2009. 7

[39] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 1

[40] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018. 1

[41] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 1, 7

[42] J. Mahler et al. Guest editorial open discussion of robot grasping benchmarks, protocols, and metrics. In *Transactions on Automation Science and Engineering*, volume 15. IEEE, 2018. 7

[43] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *Robotics Science and Systems (RSS)*, 2017. 1

[44] J. Mahler, F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Krger,

J. Kuffner, and K. Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. *International Conference on Robotics and Automation (ICRA)*, 2016. 7

[45] A. Makarenko, A. Brooks, and T. Kaupp. On the benefits of making robotic software frameworks thin. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007. 6

[46] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014. 7

[47] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra. Habitat: A platform for embodied ai research. *arXiv preprint arXiv:1904.01201*, 2019. 2, 7

[48] M. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993. 7

[49] E. Marder-Eppstein. move base, a ros package that lets you move a robot to desired positions using the navigation stack. 3, 6

[50] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001. 7

[51] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017. 5

[52] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 1

[53] L. Pinto, J. Davidson, and A. Gupta. Supervision via competition: Robot adversaries for learning tasks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1601–1608. IEEE, 2017. 1

[54] L. Pinto, D. Gandhi, Y. Han, Y.-L. Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision*, pages 3–18. Springer, 2016. 1

[55] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016. 1

[56] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009. 1, 2

[57] J. Rosen, D. Friedman, H. King, P. Roan, L. Cheng, D. Glozman, J. Ma, S. Kosari, and L. White. Raven-ii: An open platform for surgical robotics research. In *Transactions on Biomedical Engineering*. IEEE, 2012. 7

[58] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 7

[59] I. Sucan, M. Moll, and L. Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4), 2012. 6

[60] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. 2, 7

[61] E. Tsardoulias and P. Mitkas. Robotic frameworks, architectures and middleware comparison. *arXiv preprint arXiv:1711.06842*, 2017. 6

[62] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR. 7

[63] F. Xia, A. R. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese. Gibson env: real-world perception for embodied agents. In *CVPR*, 2018. 7

[64] B. Yang, J. Zhang, V. Pong, S. Levine, and D. Jayaraman. Replab: A reproducible low-cost arm benchmark platform for robotic learning. *arXiv preprint arXiv:1905.07447*, 2019. 1

[65] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. *arXiv preprint arXiv:1810.06045*, 2018. 1

A. Code Listings

```
from pyrobot import Robot

# Construct Robot.
bot = Robot('locobot')

# Construct policy.
policy = CMP()

# Relative position for each action.
dv = 0.4         # Forward step size
dw = np.pi/2.    # Rotation step size
action_position = [[0., 0., 0.0],
                   [0., 0., -dw],
                   [0., 0., +dw],
                   [dv, 0., 0.0]]

# Set goal for policy.
policy.set_new_goal(goal)
while action != 0:
    # Get image.
    rgb = bot.camera.get_rgb()

    # Compute action.
    action = policy.compute_action(rgb)

    # Execute action.
    position = action_position[action]
    bot.base.go_to_relative(position)
```

Listing 2: Visual navigation example using PyRobot API.

```
from pyrobot import Robot

# Construct Robot.
bot = Robot('locobot')

# Set pregrasp and grasp height.
pregrasp_height = 0.2
grasp_height = 0.13

# Construct grasp model.
model = GraspModel()

# Move arm and camera to reset position.
reset_pos = [-1.5, 0.5, 0.3, -0.7, 0.]
bot.arm.set_joint_positions(reset_pos)
bot.camera.set_pan_tilt(0.0, 0.8)

# Get image.
rgb = bot.camera.get_rgb()

# Compute action.
grasp_img = model.compute_grasp(rgb)

# Convert grasp from Image space to
# robot workspace.
grasp_pose = cvt_space(grasp_img)

# Execute grasp.
# 1. Go to pre-grasp pose
pregrasp_position = [grasp_pose[0],
                     grasp_pose[1],
                     pregrasp_height]
grasp_angle = grasp_pose[2]
bot.arm.set_ee_pose_pitch_roll(
    position=pregrasp_position,
    pitch=np.pi / 2,
    roll=grasp_angle,
    plan=False,
    numerical=False)

# 2. Go to grasp pose.
grasp_position = [grasp_pose[0],
                  grasp_pose[1],
                  grasp_height]
bot.arm.set_ee_pose_pitch_roll(
    position=grasp_position,
    pitch=np.pi / 2,
    roll=grasp_angle,
    plan=False,
    numerical=False)

# 3. Grasp the object
bot.gripper.close()
```

Listing 3: Grasping example using PyRobot API.

```python
from pyrobot import Robot

# Construct Robot.
bot = Robot('locobot')

# Setup gripper, camera, arm.
bot.gripper.close()
bot.camera.set_pan_tilt(0, 0.7, wait=True)

# Move hand out of camera view.
ov_pos = [1.96, 0.52, -0.51, 1.67, 0.01]
bot.arm.set_joint_positions(ov_pos, plan=False)

# Get the point cloud(in base frame).
pts, colors = bot.camera.get_current_pcd(
                  in_cam=False)

# Compute push location, direction.
pre_push_pt, push_pt, obj_center = \
    get_push_direction(pts, colors)

# Move the gripper to pre-pushing pose
bot.arm.set_ee_pose_pitch_roll(
    position=pre_push_pt,
    pitch=np.pi / 2,
    roll=0,
    plan=False,
    numerical=False)

# Move the gripper vertically down.
down_disp = push_pt - pre_push_pt
bot.arm.move_ee_xyz(down_disp,
                    plan=False,
                    numerical=False)

# Move the gripper horizontally
# to push the object.
hor_disp = 2 * (obj_center - push_pt)
bot.arm.move_ee_xyz(hor_disp,
                    plan=False,
                    numerical=False)
```

Listing 4: Object pushing example using PyRobot API.