TriFinger: An Open-Source Robot for Learning Dexterity*

Manuel Wüthrich[1] **Felix Widmaier**[1] **Felix Grimminger**[1] **Joel Akpo**[1]
Shruti Joshi[1] **Vaibhav Agrawal**[1] **Bilal Hammoud**[2,1] **Majid Khadiv**[1]
Miroslav Bogdanovic[1] **Vincent Berenz**[1] **Julian Viereck**[2,1] **Maximilien Naveau**[1]
Ludovic Righetti[2,1] **Bernhard Schölkopf**[1] **Stefan Bauer**[1]

[1] **Max Planck Institute for Intelligent Systems**
Tübingen, Germany
`manuel.wuthrich@gmail.com`

[2] **Tandon School of Engineering**
New York University
Brooklyn, USA

Abstract: Dexterous object manipulation is still an open problem in robotics, despite the rapid progress in machine learning during the past decade. We argue that a key issue which has hindered progress is the high cost of experimentation on real systems, in terms of both time and money. We address this problem by proposing a novel open-source robotic platform, consisting of hardware and software, to drastically reduce the cost of experimentation. The hardware is inexpensive yet highly dynamic, robust, and capable of complex contact interaction with external objects. The software allows for 1-kilohertz real-time control and performs safety checks to prevent the hardware from breaking. These properties enable the platform to run without human supervision. In addition, we provide easy-to-use C++ and Python interfaces. We illustrate the potential of the proposed platform by performing an object-manipulation task using an optimal-control algorithm and training a learning-based method directly on the real system.

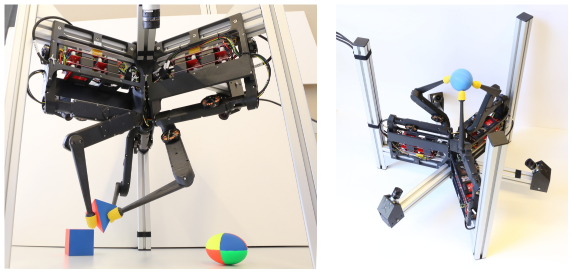

(a) (b)

Figure 1: (a) The proposed platform can be used for object manipulation on the table. (b) The same platform can be flipped and used for e.g. catching and throwing.

1 Introduction

We believe that a key issue which has hindered progress in robotic manipulation is the high cost of experimentation. Robots are typically very expensive and can break easily when entering into contact with the external world. Furthermore, they are usually operated only in the presence of a human supervisor ready to hit the emergency stop in case something goes wrong. These factors have largely prohibited systematic large scale experimentation on physical manipulation systems thus far.

*A much more detailed version of this paper can be found at https://arxiv.org/abs/2008.03596.

A large part of the robotic reinforcement-learning (RL) community has hence focused on simulation experiments [e.g. 1, 2, 3, 4, 5, 6, 7]. However, results obtained in simulation experiments do often not translate to real systems [see e.g. 8, 9]. The physics of contact interaction are nonsmooth and the outcome is highly sensitive to parameters and initial conditions (e.g. a slight difference in the shape of two objects in contact can lead to very different motions).

Therefore, we believe that an experimental platform, capable of generating large amounts of data from a wide range of possible contact interactions with external objects at a low cost, could greatly support progress in autonomous robotic manipulation.

The goal of this paper is to take a step in this direction. We present an open-source robotic platform called TriFinger. Its hardware and software design provide it with the following key strengths:

Dexterity: The robot design consists of three fingers and has the mechanical and sensorial capabilities necessary for complex object manipulation beyond grasping.

Safe Unsupervised Operation: The combination of robust hardware and safety checks in the software allows users to run even unpredictable algorithms without supervision. This enables, for instance, training of deep neural networks directly on the real robot.

Ease of Use: The C++ and Python interfaces are very simple and well-suited for reinforcement learning and optimal control at rates up to 1 kHz. For convenience, we also provide a simulation (PyBullet) environment of the robot.

Viability: The hardware design, based on [10], is very simple and inexpensive (about 5 000 $ for the complete system), such that as many researchers as possible will be able to build their own platforms. All the information necessary for reproducing and controlling the platform is open-source[1].

In the remainder of the paper, we describe the design of the hardware and software in detail. Finally, we present experiments in learning and optimal control to illustrate the aforementioned capabilities.

2 Related Work

The lack of standardized real-world benchmarks has been recognized by the robotics and reinforcement learning community [11, 12, 13, 14, 15, 16]. Recently, there have been renewed efforts in this direction: For instance, Pickem et al. [17] propose the ROBOTARIUM, a real-world benchmark for mobile robots, and Grimminger et al. [10] propose an open-source quadruped.

For manipulation, there are a number of affordable robots, such as Franka Emika, Baxter, and Sawyer. Yang et al. [18] propose Replab, a simple manipulation platform which has an even lower cost. Similarly, CMU proposed LoCoBot, a low-cost, open-source platform for mobile manipulation. Büchler et al. [19] propose a design using pneumatic actuators and show its suitability for robotic learning. However, all of these platforms have very simple end-effectors, typically 1-D grippers, which limit the possibilities of interaction with the environment.

For dexterous manipulation, there are a number of robotic hands on the market (e.g. BarrettHand, Shadow Hand, Schunk Hand) which cost typically at least 50 000 Euro per hand (an affordable exception is the Allegro hand). In addition, Dollar and Howe [20], She et al. [21], Xu and Todorov [22] proposed some innovative, exploratory hand designs. However, none of these hands are designed for long-term unsupervised operation. In addition, to have a sufficient workspace for manipulation, they have to be mounted on a robot arm, which increases the complexity and risk of damage even further.

The two setups which are most similar to the TriFinger are the D'Claw, a three-fingered robotic hand [23] and the Phantom Manipulation Platform, consisting of three Phantom Haptic Devices [24], see figure 2. As the TriFinger, both of these setups consist of three manipulators with 3 DoF each. However, the workspace, where these manipulators can interact with objects, is much larger for the TriFinger (see figure 4(a)). In addition, the Phantom Manipulation Platform is at 30 000$ about six times more expensive than the proposed setup. The D'Claw robot, at 3 500$, is in a similar price range as the proposed setup, but its actuators allow for far less dynamic motion and are less robust to

[1] https://sites.google.com/view/trifinger

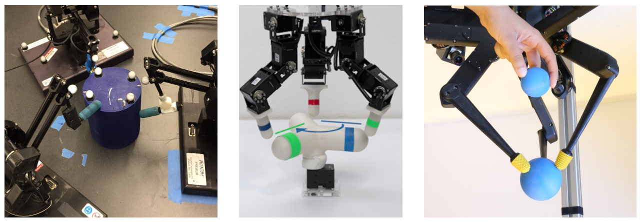


Figure 2: Phantom Manipulator (left, image from [24]), D'Claw (middle, image from [23]) and the proposed TriFinger (right). All platforms consist of three manipulators with 3 DoF each.

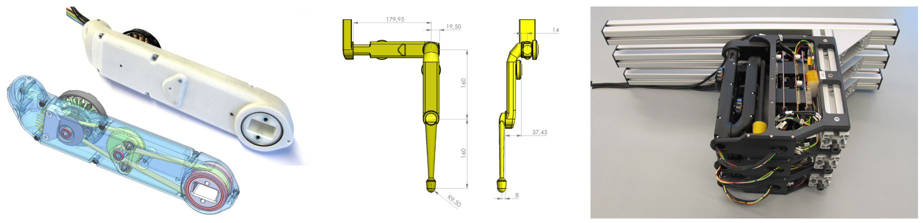
impacts, as they are only backdrivable with substantial force, i.e. they do not give in as easily. We provide more details on these points in appendix A.

3 Hardware Design

The hardware design is loosely inspired by the thumb, index and middle finger of a human hand, see figure 2. We will therefore refer to the individual manipulators as fingers and to the whole setup as TriFinger. In figure 1(b) we can see the main components of the robotic platform, including the fingers, the frame and the three cameras. In the following we will describe each of these components. All the details necessary for building an instance of the proposed platform are open-source (footnote 1).

3.1 Finger Mechanics and Electronics

The mechanics and electronics of the proposed robot are based on a recently published open-source quadruped [10] consisting of inexpensive high-performance motors, off-the-shelf parts, and 3D printed shells (see figure 3(a)).



(a) (b) (c)

Figure 3: (a) Figure from [10] showing the actuator module which is the main building block of their quadruped legs and our fingers. (b) Technical drawing of a single finger, dimensions are in mm. The size of a finger is roughly three times the size of a human index finger. (c) The platform can be disassembled into three modules for transportation and storage.

To obtain a 3 DoF finger, we add an additional DoF to the 2 DoF leg of the quadruped and mount it on a fixed basis, with some slight modifications to the 3D-printed shells and the end-effector.

We hence inherit all the favorable properties of the design from Grimminger et al. [10]:

- The high-performance brushless DC motors provide **high-torque** actuation while having a **low weight**, which allows for dynamic manipulation of objects up to a few hundred grams in weight.
- Transparency of the transmission enables **force control and sensing** and **robustness to impacts**, both of which are crucial for robotic manipulation. Transparency means that forces applied at the end-effector directly translate to torques at the motors, rather than being absorbed by a high-gear-ratio transmission. This implies that end-effector forces can be obtained by measuring the motor currents and that impacts will not break the transmission.

- The motors can be **controlled at high frequency (1 kHz)** from a consumer computer with a realtime-patched Ubuntu. This allows the robot to sense external forces and react to them extremely quickly.
- The **design is very simple** and conists of **inexpensive** off-the-shelf parts and 3D printed shells. This will allow other researchers to build their own platforms.

A simple but important addition in our design is a soft tip which mimics the human finger tip. This increases the stability of interaction with external objects greatly, as impacts are damped and contact extends to surface rather than a single point.

3.2 Kinematics

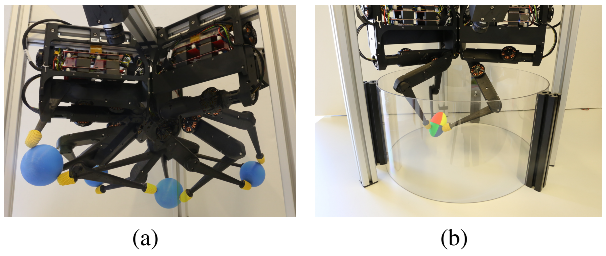


Figure 4: (a) Multiple images overlaid to illustrate the workspace. (b) A boundary to prevent objects from leaving the workspace.

The kinematics were designed to maximize the workspace where all three fingers can interact with an object simultaneously, see figure 4(a).

Each of the manipulators has 3 DoF, which implies that the finger tip can move in any direction (see figure 3(a) for technical drawing). This is important for dexterity and it makes the finger robust to impacts, as it can give in to forces from any direction.

3.3 Frame and Boundary

We use an aluminium frame to attach the fingers and cameras. The height of the fingers can be adjusted easily according to the requirements of a specific task. In addition, the entire platform can simply be flipped (see figure 1(b)) for tasks such as throwing or catching.

For manipulation on the table, we designed an optional boundary to confine objects to the workspace of the platform, see figure 4(b). This is essential for learning during extended periods of time without human supervision.

Finally, the platform can be disassembled easily into three compact modules, see figure 3(c).

3.4 Cameras

As there are three fingers interacting closely with the target object, there is a lot of potential for occlusion. Therefore we place three cameras around the platform (see figure 1(b)), ensuring that the object will at all times be visible from at least one camera. We use Basler acA720-520uc cameras with Basler C125-0418-5M-2000034830 lenses, as they have a high rate of up to 525 fps using global shutter, low latency, and an appropriate field of view (see figure 5).

3.5 Measurements and Control Signals

As in [10], the signals are transferred between the control computer (standard consumer PC) and the motors through CAN at 1 kHz. In accordance with RL terminology, we will call the control signal action and the measurments observation.
Action: The input, i.e. action of this platform is a nine-dimensional vector, i.e. a desired torque for

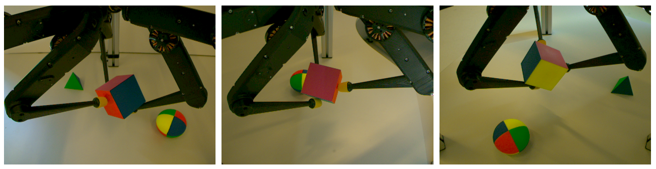


Figure 5: Images taken from each of the three cameras. Having three cameras ensures that the object will always be visible from at least one camera.

each motor. The robot expects this signal to be sent at a rate of 1 kHz.
Observation: The ouptput, i.e. the observation, consists of proprioceptive measurements (joint angles, joint velocities, joint torques) acquired at 1 kHz and images from the three cameras obtained typically at 100 Hz (rates up to 525 Hz are possible but usually not necessary).

4 Software Design

The key strengths of our software framework are:

- The user interface in C++ and Python is very **simple**, yet well-suited for real-time (1 kHz) optimal control and reinforcement learning.
- It performs **safety checks** to prevent the robot from breaking. This frees the user from this burden and allows them to execute even complex and unpredictable algorithms without surveilling the platform. This opens, for instance, the possibility of training a deep neural network policy during several days directly on the robot.
- A synchronized **history of all the inputs and outputs** of the robot is available to the user and can be logged.
- The software is designed such that new robots and simulators can easily be integrated. This may facilitate reuse of algorithms across robots.

4.1 Control Modes

There are two modes of control supported by our software:

Definition 4.1 (Real-time control)**.** By real-time control we mean that actions have to be sent to the system at a fixed rate of Δ seconds.

Definition 4.2 (Non-real-time control)**.** By non-real-time control we mean that a change in the time at which actions are applied does not change the outcome, and that actions may take varying amounts of time.

An important feature of our software design is that both modes are supported through the same interface, which makes it easy to run the same code in simulation and on the real robot.

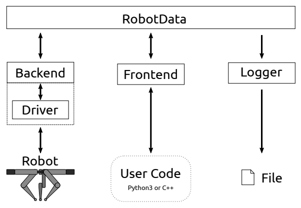
4.2 Overview

The software framework has three main components (see figure 6(a)):

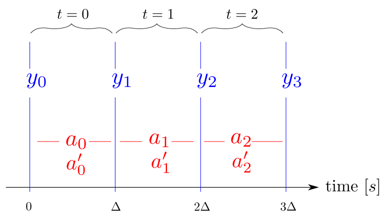
- The **back-end** communicates with the robot through the driver,
- the **front-end** allows the user to control the robot through C++ or Python3,
- the **logger** logs all the inputs and outputs of the robot.

Each of these components can run in a separate process, which is advantageous for computational reasons and it separates the back-end from the user code. Since all the commands sent to the robot flow through the back-end, we can implement checks which ensure safety no matter what happens in the user code.

In the following we will describe the front-end. The back-end will be documented in the code repository, as it is only relevant for users interested in integrating a new robot into the software framework.



(a)



(b)

Figure 6: (a) Software Architecture. The modules communicate through the RobotData and do not have any direct connections. (b) This figure shows the temporal sequence of variables. In real-time mode (4.1) there is a fixed control and observation rate Δ. Each observation y_t corresponds to a single instant in time $t\Delta$, while each action a_t corresponds to a time interval $[t\Delta, (t+1)\Delta)$.

4.3 Front-end

In our design, the only objects that exist from the user-perspective are

- a time-series of desired actions a computed by the user,
- a time-series of actions actually applied to the robot a', which is identical to a except for potential modifications to satisfy safety constraints,
- and a time-series of observations y.

Figure 6(b) shows the temporal relations of these variables. The user may perform two operations: They may append actions to the desired-action time-series a and they may read from any of the three time-series a, a', y, where a', y are filled-in by the back-end.

The user interface implementation is equivalent to the following pseudo code:

```
def append_desired_action($x$) :
    $a \leftarrow (a, x)$
    return len($a$)-1
def get_observation($t$) :
    wait until len($y$)>$t$
    return $y_t$
def get_desired_action($t$) : as above
def get_applied_action($t$) : as above
```

This interface provides access to a synchronized history of all the inputs and outputs of the robot. The user has complete freedom how to use this data and when to append actions, as long as they make sure to do so on time, before the action is needed by the robot. For instance, they may choose to run a typical real-time control loop where a new action a_t is computed periodically, or they may choose to compute entire action sequences at a lower rate and then append them in a burst by repeatedly calling `append_desired_action`.

If the user attempts to access a future observation through `get_observation(`t`)`, this function will wait and return as soon as this observations is acquired. For instance, in the case of real-time control (definition 4.1), if the call `get_observation(2)` is made at time $< 2\Delta$ seconds, the function will wait until the observation y_2 is received at time 2Δ seconds and then return. This feature allows for synchronization with the real system.

The function `append_desired_action(`x`)` will append action x to the action time-series. For convenience it returns the timeindex of the appended action, but if the user keeps track of timeindices externally, the return value can be ignored. The instant of the first call to `append_desired_action` marks time 0, after which the back-end will start filling in the a', y timeseries and expect the action timeseries a to be filled in by the user.

A basic control loop can be written as follows:

Example 4.1 (Basic control loop)**.**

```
robot.append_desired_action($a_0$)
for $t$ in $(0, ..., T)$:
    $y_t$ = robot.get_observation($t$)
    $a_{t+1}$ = some_policy($y_t$)
    robot.append_desired_action($a_{t+1}$)
```

No explicit wait is necessary, synchronization with the back-end is ensured through the call to `robot.get_observation(`t`)`, which will wait until the back-end has appended y_t.

This control loop is valid for both real-time (4.1) and non-real-time (4.2) control:
If the back-end is running in **real-time mode**, it will add observations y_t periodically and the loop above will hence run at a fixed rate, unless the call `some_policy` is too slow. If that is the case, the back-end will detect that the user did not append the next action on time and it will shut down the robot and raise an error. Alternatively, the back-end can be configured to simply repeat the previous action if the next action has not been appended on time.
In contrast, if the back-end is running in **non-real-time mode**, it will wait for the next action and the function `append_desired_action` may be called with arbitrary delay. Similarly, the execution of actions may also take varying amounts of time, and hence the call to `get_observation` will not return at a predetermined time. This mode makes sense e.g. for simulation, where the simulator and the policy add actions and observations whenever they are done with their respective computations.

4.4 Safety Checks

While the robot hardware is very robust, some additional software safety checks are necessary for ensuring that the user code cannot break the robot: We prevent collisions with the electronics, overheating of the motors and excessive velocities. Further, we shut down the robot in case there are unexpected delays, please see appendix B for more details.

4.5 Relation to Robot Operating System (ROS)

The core software described above is independent of ROS. We use catkin (which can be installed without ROS) for compilation. Further, we use ROS in some of the robot-specific packages for peripheral purposes, such as locating other packages. Finally, we use Xacro (which is part of ROS) for defining the URDF robot model of the TriFinger.

5 Experiments

The purpose of this section is not to improve the state-of-the art in robotic manipulation, but rather to illustrate the capabilities of the proposed hardware and software. Each experiment highlights different aspects:

- Section 5.1 demonstrates the 1 kHz real-time torque-control abilities and the ease-of-use of the software interface for classical control loops.
- Section 5.2 shows that the backend safety features allow for deep reinforcement learning from scratch, without any safety checks on the user side. This experiment also shows that the hardware is robust against collisions, which will necessarily occur during the learning of manipulation tasks. In addition, this use case shows that the software interface allows for application of out-of-the-box implementations of deep RL methods.
- In section 5.3, we perform a throwing experiment to show that the actuators allow for highly-dynamic motions.
- In section 5.4 we show through demonstration experiments that the platform is capable of fine-manipulation.
- Finally in appendix C.3 we discuss some experiments assessing the durability of the design.

Videos of the experiments are available at footnote 1.

5.1 Optimal Control

Controlling robot interactions with the environment is challenging due to the unilateral nature of the contact constraints and the stiff behavior of contact forces. We tackle this setting by formulating an optimization problem which yields the optimal feasible contact forces to be applied to the object, see appendix C.1. These forces can then be translated into torque commands using Jacobian transpose control.

We apply this methodology to two tasks: Lifting a cube 20 cm along a vertical line and sliding that same cube in a circle along the table. As can be seen in the videos at footnote 1, the forces applied to the object are appropriate for moving it along the desired trajectory without slippage.

5.2 Reinforcement Learning

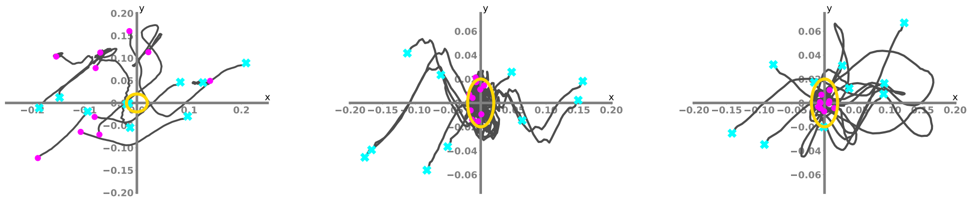


Figure 7: Trajectories of the finger tip relative to the goal position in the xy-plane. Each trajectory corresponds one episode (start: cyan cross, end: pink dot). The yellow ellipse marks 2 cm position error on each axis. From left to right: Beginning of training, after 200 episodes, end of training.

We illustrate the suitability of the platform for deep reinforcement learning by training a DDPG [25] agent from scratch on a reaching task, using the DDPG implementation from stable-baselines [26]. In this task, the goal is for each finger-tip to reach a randomly-sampled target location as accurately as possible, see appendix C.2 for more details.

We train the system for 700 episodes, corresponding to 23 minutes of execution on the robot, plus a few minutes of computation time used by DDPG. At the beginning of training, the fingers' motions are jerky, and they often collide, see the video at footnote 1. As the training progresses, the motion becomes smoother and more accurate, see figure 7. At the end of training, the fingers are able to reach the target positions consistently within an error of about 2cm.

5.3 Throwing

To showcase the ability of performing highly dynamic tasks, we execute throwing motions recorded through kinesthetic teaching (i.e. the motion was demonstrated by guiding the robot fingers). The videos at footnote 1 show that the TriFinger is able to throw light objects several meters. We expect that using appropriate controllers, rather than kinesthetic teaching, one could improve considerably on these results.

5.4 Fine Manipulation

Finally, to illustrate the dexterity of the platform, we perform several fine manipulation motions, which, as above, were demonstrated through kinesthetic teaching. As can be seen in the videos at footnote 1, the experiments include flipping a cube, turning it with one finger while the others hold it, balancing a flat cuboid on its side, and picking up a pen and drawing.

6 Conclusion

We presented an open-source robotic platform with novel hardware and software. We have illustrated through experiments its i) capabilities for dexterous manipulation, ii) suitability for deep RL from scratch thanks to the robustness of the hardware and safety checks of the software and iii) ease of use, allowing deep RL implementations to be executed out-of-the-box.

We hope that these factors, in combination with the simple and inexpensive hardware design, will make this an attractive platform for developing and benchmarking dexterous-manipulation algorithms.

References

[1] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. Jan. 2018.

[2] S. Fujimoto, H. van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. Feb. 2018.

[3] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller. Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. Apr. 2017.

[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 2016. PMLR.

[5] N. Heess, T. B. Dhruva, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. Ali Eslami, M. Riedmiller, and D. Silver. Emergence of Locomotion Behaviours in Rich Environments. July 2017.

[6] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[7] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[8] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[9] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[10] F. Grimminger, A. Meduri, M. Khadiv, J. Viereck, M. Wüthrich, M. Naveau, V. Berenz, S. Heim, F. Widmaier, J. Fiene, A. Badri-Spröwitz, and L. Righetti. An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research. In *International Conference on Robotics and Automation (ICRA)*, 2020.

[11] S. Behnke. Robot competitions-ideal benchmarks for robotics research. In *Proc. of IROS-2006 Workshop on Benchmarks in Robotics Research*. Institute of Electrical and Electronics Engineers (IEEE), 2006.

[12] F. Bonsignorio and A. P. del Pobil. Toward Replicable and Measurable Robotics Research [From the Guest Editors]. *IEEE robotics & automation magazine / IEEE Robotics & Automation Society*, 22(3):32–35, Sept. 2015.

[13] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set. *IEEE robotics & automation magazine / IEEE Robotics & Automation Society*, 22(3):36–52, Sept. 2015.

[14] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in Manipulation Research: The YCB Object and Model Set and Benchmarking Protocols. Feb. 2015.

[15] F. Amigoni, E. Bastianelli, J. Berghofer, A. Bonarini, G. Fontana, N. Hochgeschwender, L. Iocchi, G. Kraetzschmar, P. Lima, M. Matteucci, P. Miraldo, D. Nardi, and V. Schiaffonati. Competitions for Benchmarking: Task and Functionality Scoring Complete Performance Assessment. *IEEE robotics & automation magazine / IEEE Robotics & Automation Society*, 22(3): 53–61, Sept. 2015.

[16] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta. PyRobot: An Open-source Robotics Framework for Research and Benchmarking. June 2019.

[17] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt. The robotarium: A remotely accessible swarm robotics research testbed. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1699–1706. IEEE, 2017.

[18] 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.

[19] D. Büchler, H. Ott, and J. Peters. A lightweight robotic arm with pneumatic muscles for robot learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4086–4092, 2016. doi:10.1109/ICRA.2016.7487599.

[20] A. M. Dollar and R. D. Howe. The highly adaptive sdm hand: Design and performance evaluation. *The international journal of robotics research*, 29(5):585–597, 2010.

[21] Y. She, C. Li, J. Cleary, and H.-J. Su. Design and fabrication of a soft robotic hand with embedded actuators and sensors. *Journal of Mechanisms and Robotics*, 7(2), 2015.

[22] Z. Xu and E. Todorov. Design of a highly biomimetic anthropomorphic robotic hand towards artificial limb regeneration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3485–3492. IEEE, 2016.

[23] M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar. Robel: Robotics benchmarks for learning with low-cost robots. *arXiv preprint arXiv:1909.11639*, 2019.

[24] K. Lowrey, S. Kolev, J. Dao, A. Rajeswaran, and E. Todorov. Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system. In *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 35–42. IEEE, 2018.

[25] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[26] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

Appendix A Comparison with D'Claw

Here we provide a more detailed comparison between the actuator module [10] used in the TriFinger and the D'Claw actuators. The key specifications of the two robots are given in this table:

	Gear Ratio	Speed [rpm]
D'Claw	212.6:1	77
TriFinger	9:1	416

More details about the TriFinger motor can be found on the site of the manufacturer [2] (note that the values above are obtained from the motor specifications and the gear ratio of the transmission) and more details for the dynamixel module used in D'Claw can be found on the site of Robotis [3].

Hence, the maximum speed of the TriFinger joints is 5.4 times faster, which allows for more dynamic motions. Further, the gear ratio of the D'Claw is 23.6 times higher. A higher gear ratio leads to more friction in the transmission and a higher motor inertia seen at the joint level (since the rotor has to rotate 212.6 times faster than the joint). This leads to more resistance of the joint to external forces, which implies large internal forces on the transmission and hence increased risk of breakage.

Appendix B Safety Checks

There are five main checks we perform:

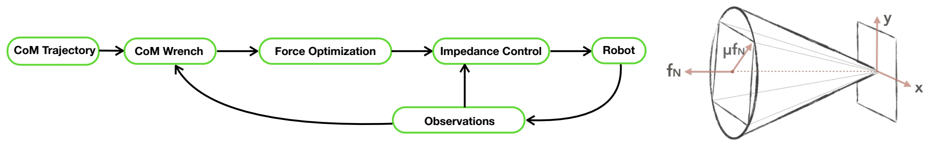
- The backend continuously monitors the timing of received actions in a real-time loop. If the expected rate of 0.001s is exceeded substantially, the robot is shut down.
- There is an additional time-out on the motor board. For instance, in case the computer crashes and the motor board does not receive any messages for some time it will shut down the motors.
- If a joint exceeds a predefined angle, it is brought back into the admissible range using a PD controller. This prevents e.g. collision of the fingers with the electronics.
- We determined the maximum admissible current to prevent overheating of the motors, and we ensure that this current is not exceeded by clipping the desired torque if necessary.
- We simulate joint damping (D-gain) to ensure that the fingers do not reach excessive velocities.

Note that we do not prevent collisions (except with the electronics), since the hardware design is robust to collisions. In addition, the software design is such that users may easily implement their own robot or even task-specific safety checks.

Appendix C Experiments

C.1 Optimal Control

We implement the control loop depicted in figure 8(a). The center of mass wrench is computed such



(a) (b)

Figure 8: (a) Control loop at 1kHz with force optimization. (b) Linear approximation of the friction cone: $|f_p| \leq \frac{\mu}{\sqrt{2}} f_n$, where f_n, f_p are the force components normal and parallel to the contact surface, respectively, and μ is the static friction coefficient.

[2] https://store-en.tmotor.com/goods.php?id=438

[3] http://www.robotis.us/dynamixel-xm430-w210-r/

that it attempts to correct the error between the current object position and its desired position along the trajectory. The challenging question is how the three fingers can apply forces to the object such that this wrench is produced, without slipping or losing contact. This can be achieved by formulating a quadratic program to find the minimal finger tip forces which achieve the desired wrench and lie within an approximation of the friction cone, see figure 8(b). These forces can then be translated into torque commands using Jacobian-transpose control.

C.2 Reinforcement Learning

The goal is for each finger-tip to reach a randomly-sampled target location as accurately as possible. The episode length is set to 2 seconds. The observation space of the policy consists of the joint positions, the joint velocities, and the target positions for each finger specified in task space. The action is the desired joint configuration of the fingers (the software backend provides a PD controller). The reward at each time step is the negative Euclidean distance between the end-effector and the target.

C.3 Durability Experiments

We ran durability experiments on a single finger, executing a fast motion in free space. In the first experiment a timing belt broke after 79 days of continuous operation. This may be partially due to the nonuniform stress put on the timing belt by such repetitive motions, which would be less of an issue in realistic operation. In the second experiment the shell of the center link broke after 72 days of continuous operation, the design has been improved since to avoid such breakage.

In addition, we performed some tests on a TriFinger version used in a robot competition hosted at our institute[4]. That version, called TriFingerPro, was developed for internal use and is too complex for open-sourcing. Nevertheless, it is essentially identical in terms of kinematics and actuation, and we would expect its durability to be indicative of the open-source version. We executed random motions, including collisions between fingers and with external objects, on two of those platforms for one week continuously without breakage.

These results are naturally not statistically significant, and the durability most likely depends on the material used for the 3D printing. Nevertheless, they are promising and we believe that we can further improve durability by fixing weak points in the design as they emerge.

[4]`https://real-robot-challenge.com/`