Behavior Cloning for Mobile Manipulation

Marcel Torne Villasevil
marcelto@mit.edu

Abstract

We envision a future where everyone will have home robots in their place. These will be able to assist humans in all sorts of tasks for example helping with cooking and cleaning. We believe these robots must have a combination of three main capabilities: navigation, object manipulation, and reasoning. In this project, we designed a simple real-world environment where a robot has to find a randomly placed block in a room and pick it up. All of the learning is done in a real-world environment, nothing is done in simulation. We learn the controller using Behavior Cloning on expert trajectories collected from a programmed expert script controller. The robot controlled by the learned script successfully finds and picks up the block 75% of the time, the same success rate as the expert script. [1]

1 Introduction

Robots can improve the lives of millions of people. Among many other applications, we envision robots helping impaired people in their everyday life, for example by assisting with house chores, or other tasks for which they now need other human help to execute them. This is just one example among many where robots will enhance human activities. Robots will also save lives, for example by executing rescue missions more efficiently than humans and without risking human lives. The success of this project brings this reality closer.

In this paper, we tackle the problem of Mobile Manipulation, where a mobile robot with an arm executes some basic tasks around a room. More concretely, we design a controller for the robot to find a block randomly placed in the room and pick it up. We decide to do all of the training in real life, without the assistance of any simulator. There are two main reasons for this, first, when learning in simulation we face the problem of sim2real transfer. Second, we are convinced that as for humans, robots should be able to learn uniquely in the real world from relatively few data samples. Hence, restricting the experiments and learning in the real world will force us to solve problems which will take us closer to finding less data-intense solutions and closer to how humans learn.

Learning the controller uniquely in the real world is a challenge because the main technique used nowadays is Reinforcement Learning. However, these methodos are very data-intensive and would take too long to run in the real world. For this reason, we learned the controller using a Behavior Cloning algorithm on expert trajectories generated by a script that we designed. You might ask yourselves, why are we learning a controller if we can already design a script that solves this task. The answer lies in that this is just a first test experiment. Knowing that we succeeded in learning from this expert agent, the next steps after this paper are to scale this up to solve more complex tasks by collecting the expert data from humans teleoperating the robot.

2 Prior Work

Sun et al. [2021] already developed a similar experiment where they studied "how robots could autonomously learn skills that required a combination of navigation and grasping". Very similarly to

[1] The code for this paper can be found at: `https://github.com/Improbable-AI/locobot-learn.git`

6.484 Computational Sensorimotor Learning Final Project (Spring 2022).

what we have in mind, they designed an environment where the robot would navigate and would clean up the room. They also had the robot learn continuously in the real world. However, our research will differ in that we are using Behavior Cloning instead of ReLMM which is less data-intensive than the latter.

Furthermore, Ghosh et al. [2019] proposed an algorithm to make it possible to learn from sparse rewards settings such as ours by using all trajectories as a sample to learn from. The algorithm they propose is called goal-conditioned supervised learning (GCSL). All trajectories can be reused by relabelling them with the goal that was instead achieved instead of the goal the agent was supposed to achieve. We believe that this work will be useful to extend our research and learn a controller that solves more complex tasks.

Finally, Hoque et al. [2022] emphasizes the importance of learning controllers in real-world scenarios instead of simulators because of the sim2real transfer problems. For this reason, they propose a new way to perform robotics testbed experiments via the Cloud. This would greatly benefit the community by providing a standardized testbed and reducing the non-reproducibility problem that we are facing due to different setups in each lab. Even though this work is not directly related to the specific research problem we are trying to solve, it reinforces our belief in the importance of using real-world environments as opposed to simulators.

3 Methods

Recall that the final goal behind this project is to design a controller for a robot to execute tasks in a home. Ideally, this robot would be capable of doing a wide variety of tasks from cooking to cleaning or assisting humans. In this project, we simplify these tasks to the basic task of picking up blocks in a room. We will design a learned controller for a robot to find a block in the room and pick it up.

3.1 Hardware setup

The robot we designed the controller for is the LoCoBot (Wögerer et al. [2012]) seen in Figure 2. This one consists of a kobuki base, for navigating around a flat room, a dynamixel arm, for grasping objects, and finally, a realsense2 camera for a first-person view from the robot. The realsense2 camera contains both a depth and an RGB camera. Finally, we use an additional realsense2 camera for a third-person view of the robot.

The LoCoBot has a NUC machine on board. However, this is a simple machine that does not have a GPU and has low computing power. For this reason, only the low-level controller for the robot is run on-board and the high-level controllers which are more resource-intensive are run in an off-board lambda machine. The communication between the onboard and off-board machine is performed through ROS and ROS topics Stanford Artificial Intelligence Laboratory et al.. Furthermore, for the low-level control of the robot, we use the PyRobot library Murali et al. [2019] which already supports the LoCoBot. Nevertheless, we adapted the code from the original PyRobot library to be able to run it on ROS-noetic version and Python3. This made it much easier to create the Neural Network models using the last Pytorch version and rolling them out on the robot, avoiding boilerplate code. [2]

3.2 Environment description

As explained earlier, we designed a system to learn the controller for the robot uniquely in real-time. We built a dedicated space, seen in Figure 1, where we perform the experiments with the robot, we call it the arena. The arena is surrounded by a 20cm wall made of wood planks. This makes it easier to design an algorithm that identifies where are its boundaries. Since the robot we are using has a depth camera, we can use this to filter out walls by discriminating against points above a height threshold. Furthermore, we built a 4-meter height stand to hold a realsense2 camera to obtain a third-person view of the entire arena.

The action space of this environment will consist of five different actions: *move forward, move backward, turn left, turn right, grasp*. Furthermore, we provide multiple sets of configurations in the

[2]The modified PyRobot version to run in Python 3 can be found at: `https://github.com/Improbable-AI/pyrobot/tree/remove-orb2-slam`

Figure 1: In this Figure, we can see the arena we built to collect the data and deploy the agent in the real world. It is bounded by wood studs, these were chosen as walls since it becomes easier to delimit the boundaries of the arena when using the point cloud camera (realsense2) on the LoCoBot. We additionally built a stand for another camera to have a third-person view of the entire arena.

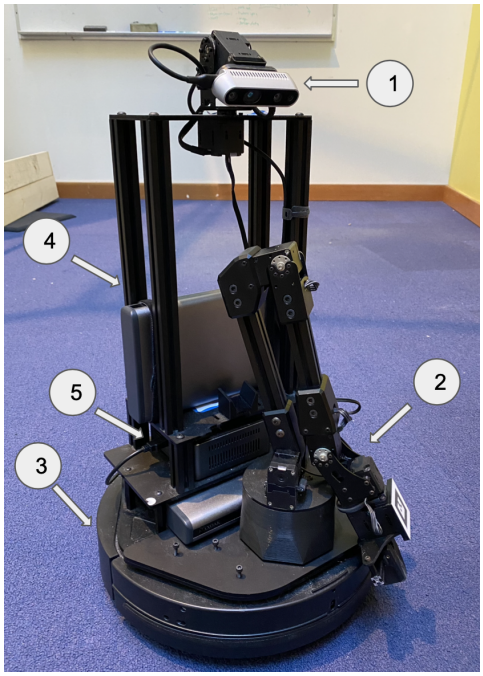


Figure 2: This is the LoCoBot robot which we run the experiments with. 1. is the realsense2 camera for the first person view. 2. Dynamixel arm. 3. Kobuki base. 4. Exchangeable Battery. 5. NUC onboard machine.

state space. We provide the first-person RGB image, the first-person depth image, the third-person RGB image, and, the third-person depth image. In our experiments, we only use the RGB images.

3.3 Behavior Cloning

The first challenging design decision we had to make was how we were going to learn the controller. One of the most popular ways is using a Reinforcement Learning algorithm such as PPO (Schulman et al. [2017]). However, these are very data consuming algorithms and since we are learning in the real world then such methods take too long and become infeasible. Another common method is to learn the controller using Behavior Cloning, this consists in imitating an expert action in a supervised learning fashion. We decided to proceed with this one.

Using Behavior Cloning algorithms implies that we need to collect data from an expert agent and then train a Deep Neural Network to perform a classification task on given the state it predicts the action to take. Recall, that the state space is an RGB image (either the first person view or third-person view of the robot), hence this will become a classification task on images. ResNet-18 has proven to perform very well on image classification tasks He et al. [2015]. For this reason, we decide to use it as an image feature extractor. The Neural Network we design for learning the controller (in Figure 3) will consist then on a feature extractor (using ResNet-18) and a sequence of fully connected layers

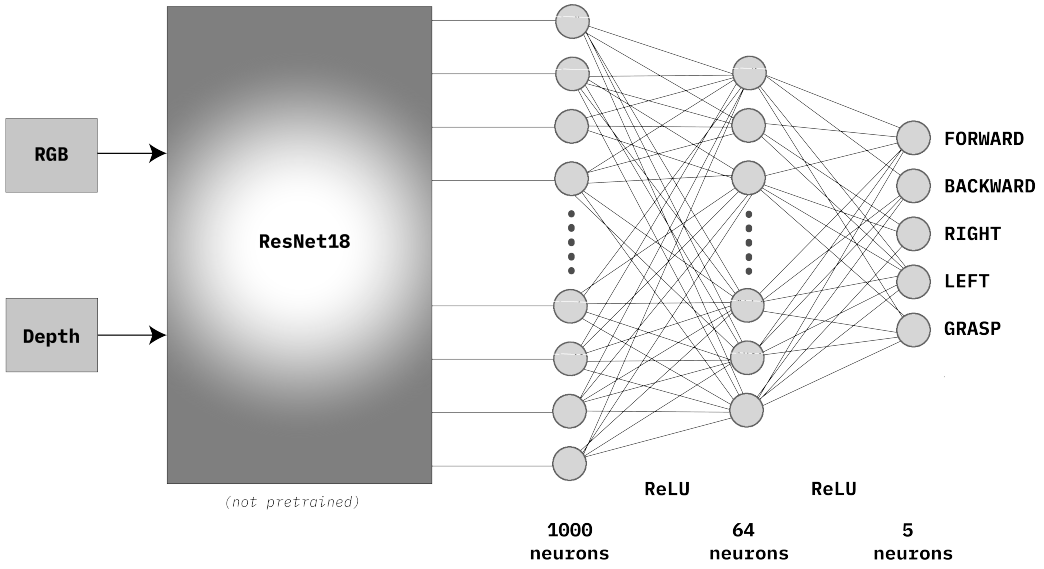


Figure 3: Deep Neural Network Architecture to learn the controller policy using Behavior Cloning. This takes as input an RGB image (first person or third person) and can also take a depth image as input. We use the ResNet-18 architecture as a feature extractor from the image input, without pretraining, and pass its output to a fully connected layer which outputs the action to take given the current state.

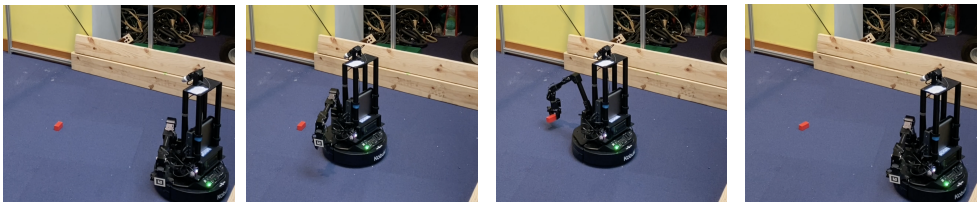
that will output the action to take. Note that we will train everything end-to-end and we will learn the ResNet-18 weights from scratch.

3.4 Expert agent

A challenge during the project was to decide which was going to be the expert agent used to collect data with. In most prior work, the expert agent is usually a human teleoperating the robot when the task is complicated. However, the task designed is simple enough that we were able to write a script to solve it with a high success rate. Using a script as an expert agent, if this one is good enough, is very helpful since it avoids many human hours of teleoperating the robot during data collection and much more data can be collected. For this reason, we decided to move forward by using a script as the expert agent to perform Behavior Cloning to learn the controller.

Algorithm 1 Expert Agent Script: get_action

```
state ← input
block_pos ← get_block_position(state)
if block_pos is None then
  return TURN_LEFT_ACTION
else
  angle, distance ← block_pos
  if angle > ANGLE_LIMIT then
    return TURN_LEFT_ACTION
  else if angle < - ANGLE_LIMIT then
    return TURN_RIGHT_ACTION
  else if distance < MAX_DISTANCE_TO_OBJECT then
    return GRASP_ACTION
  else
    return MOVE_FORWARD_ACTION
  end if
end if
```



(a) (b) (c) (d)

Figure 4: Series of frames corresponding to expert agent behavior for the first experiment task. Image a is the starting position of the robot, then the robot moves in front of the block (image b), and proceeds by picking up the block (image c). Then the environment is reset, the robot drops the block and moves backward to recover the initial position.

The designed expert agent script follows Algorithm 1. In summary, the robot rotates until the object is seen. Once the object is seen it moves forward to it, adjusting the rotation to maintain the block centered in front of the robot. Finally, once the robot is close enough to the block it executes a grasping action.

We proceed by going into more detail on the different modules. Object detection is done by performing background subtraction using the block's color. We use a red block and the background is blue, for this reason, a simple color extraction is very efficient in this scenario. A downside is that it does not generalize to objects with different colors. The grasping module we use is an implementation by Gupta et al. [2018], and we inspired our code from the code provided by PyRobot.

3.5 Experiment 1: Forward pick block

As simple as the task of finding a block in the room and picking it up might seem it is still not trivial. For this reason, before making the robot find an object at any place in the arena, we simplified the task and restrict the locations where the block is placed. We designed the environment such that the block will always be placed in front of the robot between 2 and 4 forward actions by the agent.

We will collect data by using our expert agent script to go forward and pick up the object, then we will reset the environment by forcing the robot to take 3 actions backward. This process can be seen in Figure 4. Furthermore, in this case, we will learn the controller using only the first-person RGB image view.

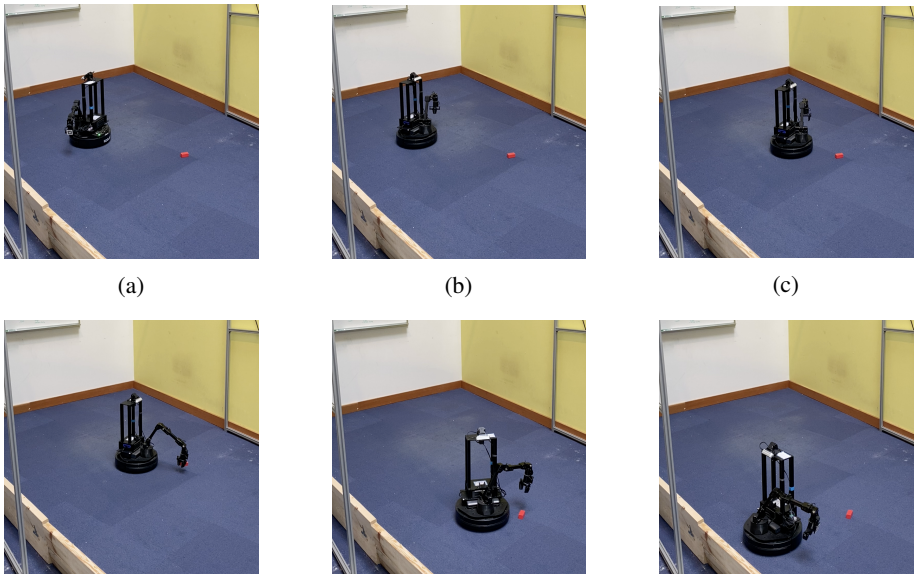
3.6 Experiment 2: Random pick and drop block

After succeeding with the first experiment, we can make the experiment harder and learn a controller to pick up the block placed anywhere in the arena. In this case, we start the experiment with the block placed somewhere in the arena and proceed by executing the expert agent script. Once the object is picked up, we reset the environment by using the robot to drop the block somewhere randomly on the arena and then move the robot somewhere else, again randomly. Note that this reset procedure is automated and does not need human supervision.

Finally, for this second experiment, we added a sixth action into the action space: *move a small step forward*. This action makes a shorter forward step than the simple move forward. Introducing this action helped us increase the performance of our expert agent script and hence increase the efficiency in data collection. The behavior of the expert agent for this experiment can be seen in Figure 5.

4 Results

We proceed by showing the promising results of both experiments, where we succeeded in learning a controller for the robot to pick up the object.



(a) (b) (c)

(d) (e) (f)

Figure 5: Series of frames corresponding to expert agent behavior for the second experiment task. *Image a* is the starting position of the robot, the robot starts without a first-person view of the block. Then the robot rotates until it visualizes the block (*image b*). It proceeds by moving forward towards the block (*image c*), and picks it up (*image d*). The environment is then reset, the robot moves to a random position to drop the block and moves again to another random position away from the block.

4.1 Data collection

As we implied earlier, the expert agent script used to collect expert trajectories is not perfect. The performance obtained collecting data for the first experiment is around 85%. And, the performance in the second experiment is around 75%. The reason why it does not achieve a 100% performance in picking up the block is because the grasping module is not perfect, sometimes the prediction of the pose is wrong and sometimes the grasping execution is not accurate enough which makes it miss the target. Furthermore, our algorithm for finding the object is not perfect either, sometimes the robot gets too close to the object and it looses sight of it. Nevertheless, since we want our learned controller to be better than the expert agent, when we perform the data collection, we are not storing the trajectories where grasping fails. In this way, we hope that our learned controller might beat the expert.

For the first experiment, we collected 161 trajectories, consisting of around 700 action-state tuples. This was collected in around 3 hours of running it on the robot. For the second experiment, we collected 167 trajectories which correspond to around 3500 action-state tuples. This was collected in around 8 hours.

4.2 Results 1: Forward pick block

We trained the first controller using the architecture presented earlier and the collected data for this experiment. More concretely, the state space in this experiment was simply the RGB image from the first-person view. In Figure 6 we observe the training loss of the model while learning. We also observe that the test loss is relatively low, around 0.3.

To test this learned model, we deployed it into the real robot and tested whether it worked properly. We present the success rates in Table 1. It is interesting to see that when the controller finds itself in a similar region to the ones where data was collected before then the performance is relatively close to the expert (70% our model against 85% for the expert). However, when the agent moves to less seen regions (mainly close to the walls) then the performance drops significantly to 30%. This is a common problem in the generalization of Behavior Cloning and supervised learning in general. When tested on out-of-distribution data the performance decreases. We believe this can be improved

by collecting more diverse data. Furthermore, it can also be improved using DAgger. Nevertheless, we leave this as future work.

	Expert Agent	Learned Agent (BC)
Familiar trajectories	85%	70%
Unfamiliar trajectories	85%	30%

Table 1: Success rates comparison between learned agent and expert agent, the learned agent performs very well in zones where it was trained. However, it performs less well in unfamiliar zones, we have an out-of-distribution data issue.

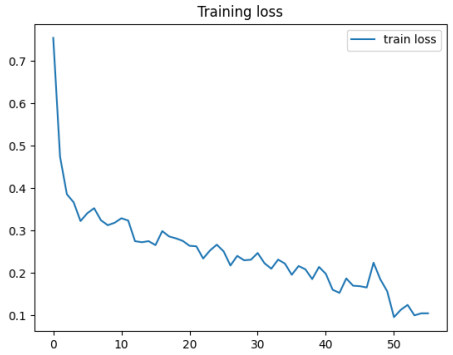


Figure 6: Training loss curve on the complex task of picking up the object placed in front of the robot on the first-person camera view. A good policy is obtained after the training which successfully solves the task with a 70% success rate.

4.3 Results 2: Random pick and drop block

We trained a second controller to solve this more complicated task. In this case, we wanted to train a controller on the third-person camera. We believe this would help the model generalize to new positions of the object since it always would have a holistic view of the arena. However, we trained this on the architecture presented earlier and we did not manage to obtain a good validation loss, as can be seen in Figure 7. The model completely overfits to the data and the validation loss never goes down. We believe that one of the reasons might be because the block is very small seen from the third-person camera and probably without enough training data the model might not put attention to it and might just take it as noise. We are still interested in making the controller work on the third-person camera. This might become useful in future work when we try to learn more complicated tasks. If the environment is not fully observable the best action can depend on previous states but if we have a fully observable state we could keep the Markovian property of the environment. For this reason, the future steps will be to collect more data.

Nevertheless, we tried changing the state space to the first-person camera and the results were much better as seen in Figure 8. This controller generalized much better than the one in the previous experiment, it attains the same performance as the expert in this task, around 75% for both the expert and the learned controller. A priori, we thought it would be much harder to learn a first-person view controller since this one does not have a fully observable state of the environment. We believe this still performed well because of the simplicity of the task, each action does only depend on the current state even without it being fully observable. Furthermore, one of the reasons we believe that this controller generalized better than the one in the previous experiment is that the collected dataset is much larger in this case. Also, the nature of the task makes the collected dataset much more diverse, since the block is randomly placed after each reset, and the robot goes further around the arena than in the first experiment where the block stays around the same place every time.

Figure 7: Training and validation loss curve on the complex task of picking up the object randomly placed in the room using the third-person camera view. The model completely overfits and no good policy is learned.

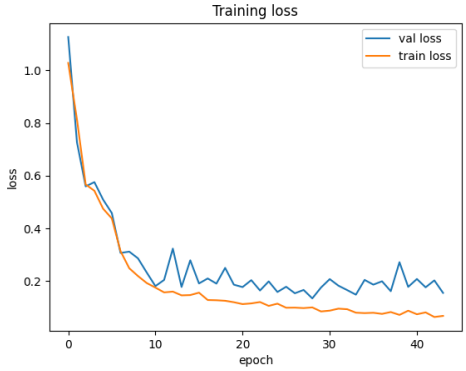


Figure 8: Training and validation loss curves on the complex task of picking up the object randomly placed in the room using the first-person camera view. A good policy is obtained after the training which successfully solves the task with a 75% success rate.

5 Discussion

To conclude, in this project, we designed an end-to-end system for learning a controller for a robot uniquely trained in the real world. We solved the final task of having a robot locate a block in an arena, move towards it and pick it up. The process described in this paper to achieve this result consisted in training a controller for the LoCoBot using Behavior Cloning on an expert agent that we programmed. We built a real-world environment, which provides two state spaces: the first-person camera and the third-person camera view. We designed and solved a subgoal before solving the final task, this consisted in restricting where the block could be placed, by just having it placed in front of the robot at a range distance. We collected data with the expert agent script and trained a model using Behavior Cloning. Once we solved this task, we repeated the same process and solved the final goal by achieving the same success rate as the expert agent, around 75% of the time the block was successfully picked up.

6 Contribution

I (Marcel) have been the main author of the paper and the work presented here. Nevertheless, this project has been done in collaboration with the Improbable AI lab. I have been supervised by Dr. Abhishek Gupta, Tao Chen, and Prof. Pulkit Agrawal. This project is a section of a bigger research project that is still under development and that I am excited to continue throughout part of the summer and during my Master's Thesis next year.

References

D. Ghosh, A. Gupta, J. Fu, A. Reddy, C. Devin, B. Eysenbach, and S. Levine. Learning to reach goals without reinforcement learning. *CoRR*, abs/1912.06088, 2019. URL http://arxiv.org/abs/1912.06088.

A. Gupta, A. Murali, D. P. Gandhi, and L. Pinto. Robot learning in homes: Improving generalization and reducing dataset bias. In *Advances in Neural Information Processing Systems*, pages 9112–9122, 2018.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

R. Hoque, K. Shivakumar, S. Aeron, G. Deza, A. Ganapathi, A. Wong, J. Lee, A. Zeng, V. Vanhoucke, and K. Goldberg. Learning to fold real garments with one arm: A case study in cloud-based robotics research, 2022. URL https://arxiv.org/abs/2204.10297.

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. *arXiv preprint arXiv:1906.08236*, 2019.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Stanford Artificial Intelligence Laboratory et al. Robotic operating system. URL https://www.ros.org.

C. Sun, J. Orbik, C. M. Devin, B. H. Yang, A. Gupta, G. Berseth, and S. Levine. Fully autonomous real-world reinforcement learning with applications to mobile manipulation. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=Esw0Wh6Stt5.

C. Wögerer, H. Bauer, M. Rooker, G. Ebenhofer, A. Rovetta, N. Robertson, and A. Pichler. Locobot - low cost toolkit for building robot co-workers in assembly lines. In *Proceedings of the 5th International Conference on Intelligent Robotics and Applications - Volume Part II*, ICIRA'12, page 449–459, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 9783642335143. doi: 10.1007/978-3-642-33515-0_45. URL https://doi.org/10.1007/978-3-642-33515-0_45.