Robot arm teleoperation via RGBD sensor palm tracking

Filip Marić, Ivan Jurin, Ivan Marković, Zoran Kalafatić, Ivan Petrović

University of Zagreb, Faculty of Electrical Engineering and Computing
Unska 3, 10000 Zagreb, Croatia
{filip.maric, ivan.jurin, ivan.markovic, zoran.kalafatic, ivan.petrovic}@fer.hr

***Abstract*—In this paper we present a simple and intuitive approach to teleoperating a 3 or higher degrees-of-freedom (DOF) robotic arm in Cartesian space. Using an RGBD camera, we retrieve the position of the user's palm. This position is then translated into the desired robotic arm position, which is then used as an input to a control loop. The entire system is implemented in the Robotic Operating System, enabling simple functionality transfer to any compatible robotic arm. The system was tested on the Kinova Jaco 6DOF robotic arm with the aim of using it for object manipulation. We use the inverse kinematics for calculating the joint rotation velocities required for following the Cartesian path of the human hand. The resulting joint velocities are then sent to the robotic arm control interface which then passes commands to the pertaining API. Results corroborate the validity of the proposed approach for robotic arm teleoperation, opening the possibility for many potential applications.**

***Index Terms*—Robotic arm, Palm detection, Teleoperation, Kinect**

I. Introduction

Robotic arms have been used for decades as means of manipulating objects in dangerous environments, performing arduous, repetitive tasks, and assisting the disabled. In cases where robotic arms are not autonomously performing these tasks, they are usually teleoperated by humans. More often than not, these methods of teleoperation offer a somewhat steep learning curve, requiring the user to be familiar with various modes of control and settings. Some newly developed methods [1]–[3] are more intuitive, but require the user to wear certain equipment which is not always practical and requires a sizeable amount of set up time. On another note, a common problem with machine learning systems like [1], [2] is that they require training for each new user since joint mapping does not scale.

Successful replication of human arm movement with robotic arms is barred by the complexity of the shoulder and finger joints. In practice most currently available robotic arms cannot match human dexterity. For this reason, direct mapping of human arm joint positions to a robotic arm is still not a viable option, but is a promising one for the future.

Such considerations have caused researchers to explore various modes of teleoperation that reduce the dimensionality of control required by the operator, giving some control up to various algorithms in exchange for a more practical system. Studies in human motor-control have suggested that a low dimensional representation is feasible at the kinematic level [4]. The work presented by [1], [2] follows this result, using electromyography signals corresponding to the activation of the shoulder muscles to control a robotic arm in 3D space. We find another interesting approach in [3], where a rig with a force feedback mechanism is used to map the joints of a human arm to an anthropomorphic robotic arm.

In this paper we propose a system which produces a simple and intuitive way of manipulating objects that, once set up, requires little to none prior knowledge of the system. The hardware used for detection is easily obtainable and low-cost. We maintain a focus on making the system generalizable to a variety of robotic arms, exposing all the important set up parameters through the Robot Operating System (ROS) parameter interface. All of the libraries and packages used in this project are well documented and maintained, hopefully making our code compatible with future versions of ROS. Unlike the majority of complex and slow algorithms used in hand-pose estimation and tracking problems, the main advantage of the proposed approach is its simplicity and speed which offers a large bandwidth for possible upgrades.

The paper is organized as follows. In Section II we cover the theoretical framework for such a system under the Robot Operating System (ROS) architecture. Section III contains an overview of the hand-tracking algorithm developed for the Microsoft Kinect RGBD camera. In Section IV we describe the proposed system for controlling the Kinova Jaco 6DOF robotic arm. The system was tested in a variety of situations, with an accent on prehensile and non-prehensile manipulation of everyday objects. It is our opinion that these results show potential for this kind of system in a number of different applications. In the end, Section V concludes the paper.

II. System Overview

Conceptually, our proposed system takes the form of a high level control loop (Fig. 1). In this analogy, the detected human hand and robotic arm positions serve as input and feedback data, respectively. Before subtraction, both the input and feedback data are placed in the same reference frame. The motion planner, serving as the regulator, subtracts the data and

This research has been partly supported by the Ministry of Science, Education and Sports of the Republic of Croatia under the grant "Centre of Research Excellence for Data Science and Cooperative Systems".

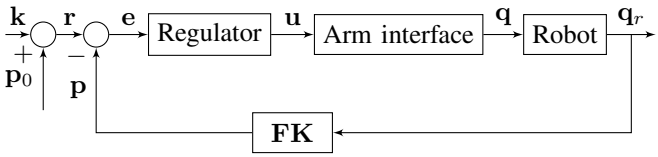

Fig. 1: Block diagram representing the system. Note that $\mathbf{FK}$ represents the forward kinematics of the robot. The control and interface blocks compose the motion planner.

converts the resulting difference into joint speeds of the robotic arm. In this section, we focus on the motion planner and the various components it utilizes to control the motion of the robotic arm.

A. Input data

The input data takes the form of a vector in $\mathbb{R}^6$ and a separate scalar in $\mathbb{R}$. The scalar c represents the amount of end-effector closure, while the values $x,\ y,\ z,\ \psi,\ \theta,\ \rho$ represent the positions and rotations in Cartesian space:

$$\mathbf{k} = \begin{bmatrix} x & y & z & \psi & \theta & \rho \end{bmatrix}^T; \quad c\ . \tag{1}$$

We denote the current pose of the end effector in the base coordinate frame as $\mathbf{p}$ and the end-effector closure as c_p:

$$\mathbf{p} = \begin{bmatrix} x & y & z & \psi & \theta & \rho \end{bmatrix}^T; \quad c_p\ . \tag{2}$$

To make the input data usable, we need to transfer it to a coordinate frame of the robotic arm. Another thing which we also have to account for is the difference in range between the human and robotic arms.

One possible solution for this would be to find a transfer matrix between the Kinect and robot arm base coordinate frames, directly mapping $\mathbf{k}$ and c to the desired pose of the end-effector. The downside to this method is that it depends on the position of the Kinect relative to the robotic arm. This makes it impractical to set up the system and makes it prone to outside disturbances. We circumvent this problem by making the system start only when the human hand is near a certain position in the Kinect coordinate frame. When the hand is first detected, its position it then matched with the starting position of the end effector $\mathbf{p}_0$. In this manner we create a new coordinate system at the center of the human hand. All movement in this coordinate system is interpreted as the displacement of the end-effector from the position $\mathbf{p}_0$. Now we can define the modified reference data $\mathbf{r}$:

$$\mathbf{r} = \mathbf{p}_0 + \mathbf{k}\ . \tag{3}$$

Now we can also define $\mathbf{e}$ as the difference vector between the current hand position and the starting position displaced by detection data:

$$\mathbf{e} = \mathbf{r} - \mathbf{p} = \mathbf{p}_0 + \mathbf{k} - \mathbf{p}\ . \tag{4}$$

Notice that the end-effector closure variable c is coordinate system independent, thus allowing us direct mapping:

$$\mathbf{c}_p = \mathbf{c}\ . \tag{5}$$

B. Regulator

The regulator uses the value $\mathbf{e}$ to control the robotic arm path. The path is realized and regulated using the Cartesian velocity interface of the robot. We have chosen this option as opposed to a position based approach since it avoids time-based parametrization and is inherently smooth in most cases. One of the advantages this gives is the ability to use PID regulation on the path itself, which determines the inertia with which the robotic arm follows the human hand.

If we interpret $\mathbf{e}$ as the unregulated desired velocity of the arm then we can define $\mathbf{u}$ as the regulated variant. The D and I functions represent the integral and derivative components, respectively [5]:

$$\mathbf{u}(k) = K_p\mathbf{e}(k) + K_i\mathrm{I}(\mathbf{e}(k), T) + K_d\mathrm{D}(\mathbf{e}(k), T), \tag{6}$$

where T represents the rate at which the system operates. It is practical to adjust T to values where the distance the robot hand can move in one interval is strictly lower than $\mathbf{u}$. This adjustment keeps the movement smooth since it is less likely that the end-effector will overshoot the target position.

In cases where a Cartesian velocity interface is not available, one can be constructed given that the robotic arm supports joint velocity commands. This requires an exposed inverse kinematics function for the particular robot which is used for transforming Cartesian positions to joint values. The inverse kinematics problem is not trivial and can be approached in a variety of ways. Given the inverse kinematics function IK and the values of current joint angles $\mathbf{q}$, we can calculate the required joint velocities:

$$\mathbf{u}_q = \mathrm{IK}(\mathbf{r}) - \mathbf{q}\ . \tag{7}$$

Another way to transform Cartesian into joint velocity would be using the inverse Jacobian method [6]:

$$\mathbf{u}_q = \mathbf{J}^{-1}\mathbf{u}\ . \tag{8}$$

This method applies to velocities, but instead we use $\mathbf{u}$ without changing any parameters. We use the fact that $\mathbf{u}$ is relatively small because of the high operating frequency, allowing for this kind of approximation. This method was tested in a simulated environment using *Gazebo*, a 3D interactive simulation system with an in-built `ros_control` interface.

The inverse kinematics of the Jaco robotic arm is efficiently calculated by the official API. Moreover, since the official API also provides Cartesian velocity commands, it was used for performing the tests described in the results section.

C. ROS implementation

ROS is an open-source, meta-operating system that handles hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools

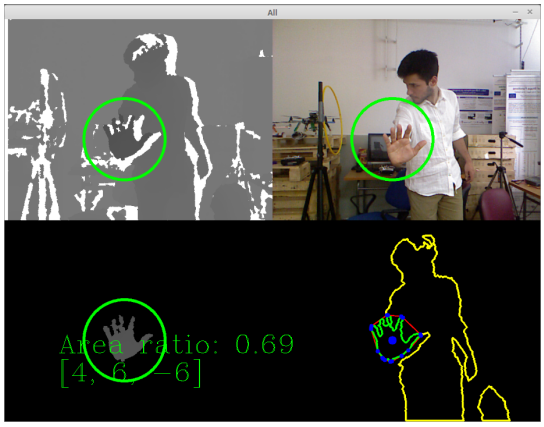

Fig. 2: Graphical user interface for hand tracking

and libraries for obtaining, building, writing, and running code supporting distributed computing. At the file system level, the main organizational component of a ROS system is the package. A package may contain executables (nodes), libraries, data-sets and configuration files. In a robot control system, nodes process data and communicate with each other through the Computation Graph.

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. Communication is done by nodes subscribing to and publishing standardized data structures called messages by way of topics. Topics can be seen as a location for a certain type of message to be subscribed and published to. Possibly the greatest advantage ROS has on other similar projects is the community. Most of the packages are community maintained by a large and active user base, which makes the process of learning the system considerably easier.

The system is implemented in ROS Indigo-Igloo, compatible with Ubuntu 14.04 OS. It consists of 2 nodes: the `hand_detection` node interfaces with Kinect and runs the hand detection algorithms on the recieved data, the `arm_control` node generates the control data $\mathbf{u}$, which is then sent to the `ros_control` interface. The `hand_detection` node, described in more detail in Section III, publishes the processed Kinect data $\mathbf{u}$ in the form of a `geometry_msgs\Twist` ROS message[] to the `detection_data` topic.

The `detection_data` topic is subscribed to by the `arm_control` node, which runs the control algorithm every time a new message arrives to the topic. From this follows that we can control the refresh rate T by adjusting the publishing rate of the `hand_detection` node.

The `arm_control` node then sends arm and gripper commands to their respective controllers, spawned and managed by the `ros_control` interface. It is important to note that the gripper is also controlled using joint velocities, so a value c of 0 would correspond to a maximum velocity closing of the gripper. This is managed by configuring the controller joint angle and velocity limits in the configuration files.

III. Hand-tracking Algorithm

The problem of hand pose estimation is already solved with some degree of success in classic RGB images using complex algorithms and large training sets [7], [8], [9]. The main disadvantages of using that approach lie in its dependency on the light parameters of the image as well as its complexity. These approaches have problems with real-time detection which is a requirement for our system. As depth cameras became more and more available to general public, an opportunity is provided to use depth data as a more reliable and robust data source which works well in any provided light condition.

A. Hand localization

Regardless of the camera being used, the first step in precise detection of a hand pose is to determine its position in the image. Various methods have been proposed for solving this problem, but most of them use some kind of a hint where the hand might be in the image. Moreover, many of the methods require users to wear a special non-reflective bracelet or some sort of marker object that can help estimate the hand location. This can vastly speed up hand isolation and offer more time to conquer hand pose estimation.

Qian et al. in their work [10] initially assume that hand is located closest to camera in relation to other objects in scene. Moreover, it requires human to wear non-reflective bracelet and uses flood-filling algorithm to extract part of image that is closer than the bracelet. Shotton et al. propose a different initial method, which is used to detect human pose in depth image [11]. It uses depth invariant features and trained random forest for binary per-pixel classification whether it is part of hand or not. Thompson et al. [12] use this method to extract one of more hands from the image. The method is extremely efficient and fast, especially on GPU but within this work there are no possibilities to use cluster of 1000 cores to train random forest for a few days so we were trying to conquer this problem using a simpler approach with still satisfying results.

In our work the background subtraction method assumes that hand is the closest object to the camera (Fig. 2 shows the user interface of the hand tracking algorithm). Unlike [10] the bracelet is replaced with simple and fast depth filter at an empirically determined depth offset of 110 mm. Figure 3 shows successful hand isolation invariant to provided palm incline even when palm is horizontally placed or closed. This simple but efficient and above all fast filter leaves more processing time to be spent in the upcoming detection stages.

If the closest point of the palm is on the middle finger, the carpal part of the palm could be omitted and also if the closest point is on the wrist, a part of forearm can be included (Fig. 4). Palm location is considered as the center of a contour that separates the palm from the background. Depth coordinate of the palm location is calculated in the proximity of that point. Although some hand poses can lead to the center point being outside of the palm itself, in practice this scenario is relatively rare and will be disposed as invalid (depth cameras have issues with detecting that kind of poses in the first place).

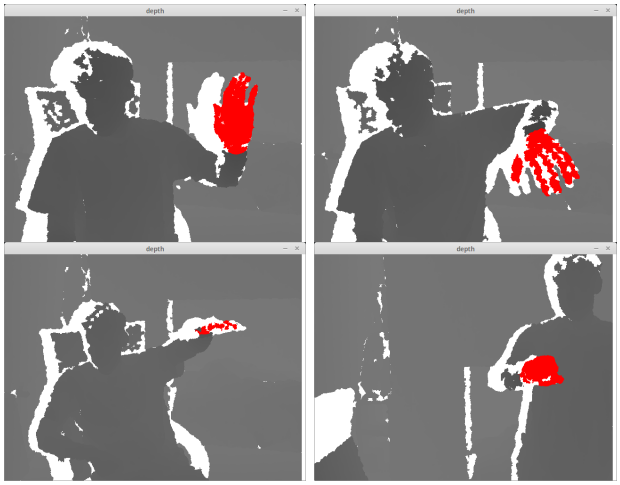

Fig. 3: Successful palm isolation

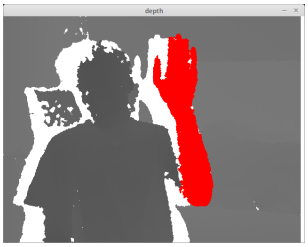

Fig. 4: Poor palm isolation

B. Hand pose estimation

Qian et al. have modeled an average hand using 48 spheres and 26 DOF [10]. Extreme points in extracted hand are candidates for tips of fingers and wrist. Using inverse kinematics they try to fit constructed model and best fit is considered as the current hand pose. Thompson et al. [12] use convolutional neural network cascade trained on set of 70 000 marked images. As well as [10] this paper also uses inverse kinematics to determine current hand pose from the neural network output. To model the hand this paper uses an open source library *libhand* [13]. It is notable to mention that both methods are able to recognize hand pose from single depth image only. Once the palm is extracted from the background, further processing is much simpler. Considering application of this tracker, precise hand pose estimation using many DOF is unnecessary. A focus has been made on palm openness as a binary indicator and some attempts to extract palm orientation have been made.

Hand openness is relatively abstract concept. We define a closed palm as a pose where every interphalangeal joint is flexed to a high degree possible. Everything else is considered an open palm. This concept emerges from the application of the tracker. As robot grasping mechanism differs from robot to robot, a direct correlation between per finger flexion and extension is not possible so the classifier determines whether robot will try to grasp something or not. This ensures the controlling does not depend on grasping type.

Fig. 5: Palm contour and convex hull

Binary classifier is a trained function of only two partly correlated features, depth invariant palm area and ratio of palm area and area of its convex hull (Fig. 5). Whilst constructing the classifier special attention has been made to prefer quality of classifying palm as closed at the expense of quality of open palm classification. This is also application specific improvement that assures once the robot has grasped something it will not release it unintentionally.

Training has been made on a database of 2600 palm images with the same amount of opened and closed examples. Using information from sequential images, a major improvement has been made by canceling noise from the depth camera. Empirically the best noise cancellation has been made using 7 sequential images and applying a mean filter on the detection outputs.

C. Results

Using an unoptimized Python script and average personal computer this method can generate up to 125 images per second (and even more if optimized and transfered to C++) and leaves space for possible upgrades. Determining the quality of detection has been accomplished in two ways. As the detector uses a single image for detection and sequential images for noise cancellation, a test has been developed for both single and sequential images. In both cases, the test consisted of classifying the palm correctly as opened and classifying it as closed.

Test has been made on 3108 sequential images of which 1323 with open and 1785 with closed palm. Unlike the learning process which was developed using a single persons palm in the testing process multiple persons have been asked to provide data. As expected, sequential images have proven to be significantly bettera with over 96% of success rate (Tables I, II, III, and IV). Closed palm detection is better than open palm detection which was the goal. The reason why the closed palm detection should be especially taken care of is the process of grasping which should be as reliable as possible. For a grasp to fail, using 30 fps data feed and having in mind robot speed, approximately 10 sequential detection fails are required. This makes it unlikely for such an event to occur.

	Hit	Miss	Sum
Example count	1203	120	1323
Percentage	90.9%	9.1%	

TABLE I: Classification of open palm state in single image

	Hit	Miss	Sum
Example count	1728	57	1785
Percentage	96.8%	3.2%	

TABLE II: Classification of closed palm state in single image

	Hit	Miss	Sum
Example count	1256	67	1323
Percentage	94.9%	5.1%	

TABLE III: Classification of open palm in sequential images

	Hit	Miss	Sum
Example count	1740	45	1785
Percentage	97.5%	2.5%	

TABLE IV: Classification of closed palm in sequential images

IV. Experimental Results

The system was developed and tested using the 6 DOF Kinova Jaco robotic arm. The official arm API offers both velocity and position commands in cartesian and joint space. This allowed us to construct a cartesian velocity interface for the robot which can handle position and orientation inputs.

A. Practical Considerations

It is useful to limit the tracking space to a sphere about 1.5 meters wide, centered at a point 1.5 meters away from the lens. This way the system is only first triggered by the opperator stepping into the sphere. Such a set up allowed for maximum movement space of the operators hand and simple cessation of operation. Leaving the sphere causes the hand to return to its starting position.

At the time of writing, an effective way of detecting human hand orientation using the Kinect is still to be developed, so the orientation of the arm remains constant in our testing. To implement a hand orientation detecting algorithm, one only has to publish `geometry_msgs\Twist` messages to the topic subscribed to by the control node. Swapping detection algorithms is made easy by use of the roslaunch system.

B. Simulation

The system proposed in Section II (Fig. 1) was simulated using the *Gazebo* robot simulation environment. As mentioned earlier, *Gazebo* allows for controlling the robot model using the `ros_control` interface. This enables the developed control algorithms to be easily transferable to a real robot, provided it also has an `ros_control` interface.

The inverse velocity kinematics for the robotic arm are calculated using *The Kinematics and Dynamics Library* (KDL) [14]. KDL uses the same robot description file as *Gazebo* to construct a kinematic chain describing the robotic arm it is then possible to solve forward kinematics, construct a Jacobian matrix and calculate inverse kinematics using iterative methods. System precision was tested by defining a

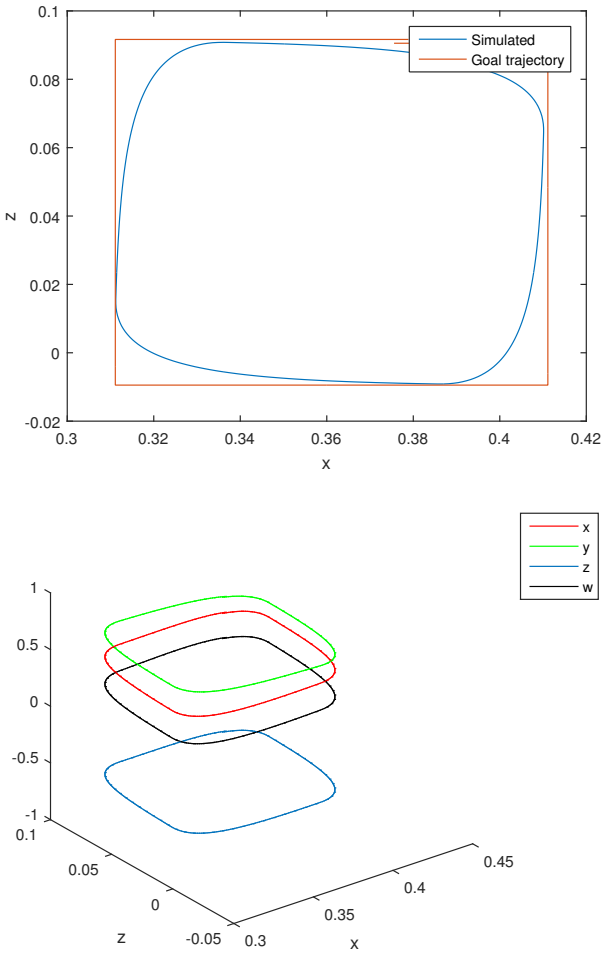

Fig. 6: **Top**: Desired Cartesian trajectory and the trajectory performed by the simulated Jaco arm. **Bottom**: Orientation of the arm during trajectory execution. In both the upper and lower images we see that the orientation is near constant.

square trajectory for the arm in the x–z plane with constant end-effector orientation. The distance between goal trajectory points is around 0.001 cm, which gives a presumed arm speed of around 3 cm/s given the Kinects refresh rate. The results show a relatively low margin of error even with the most basic form of Jacobian-based inverse velocity kinematics (Fig. 7).

C. Testing

Testing the system consisted of picking up various objects from the table and placing them in a designated area. In the first scenario, a bottle was manipulated using the arm and consecutively moved from one side of the table to the other. The second scenario consisted of placing a series of objects inside of a plastic box.

We found that a using only the proportional term of (6) gives satisfying results, following the human hand effectively in 3D-space. This simplification gives a small absolute error when the hand is static, but error build up is avoided since the

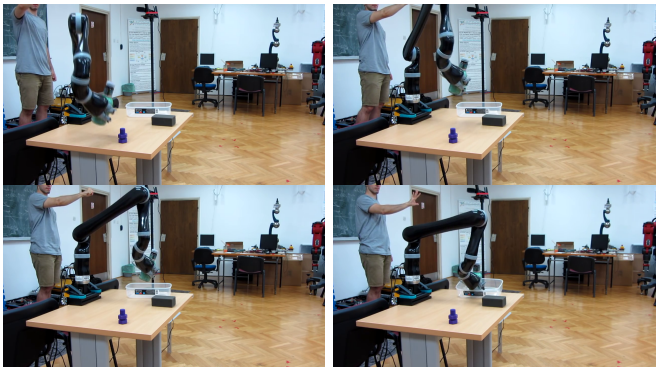

Fig. 7: Successful grasp for the first scenario

Fig. 8: Successful grasp for the second scenario

system constructs each new command using the initial position of the robot.

The first scenario, shown in Fig. 7 resulted in a successful grasp every time. We noticed that the system is suitable for grasping bottle-like objects, since the hand tracking algorithm has better hand detection capabilities when the hand is oriented towards the camera. It was also found that it is useful to limit the operation space of the manipulator in a way such that it corresponds to the free space in front of the user in order to maintain controllability. When the users hand leaves the operation space, the manipulator will return to it's initial position as defined when the system was started.

In the second scenario, shown in Fig. 8, we attempted sorting various objects into a plastic box. The lack of end-effector orientation control made it somewhat difficult to grasp smaller objects like screws, but grasping and placing non-rigid objects like sponges proved to be successful. The high refresh rate makes the system immune to noise when detecting hand closure, resulting in a very low percentage of failed grasps.

V. Conclusion

Visual teleoperation is proven to be a useful tool, but to make it usable outside of a laboratory setting, it needs to be made simple and affordable. In our research we have shown that this is possible by using relatively low-priced detection hardware and open-source software. The open-source ROS infrastructure enabled us to develop high abstraction algorithms, making the system reusable and adaptable to a variety of hardware. The hand detection algorithm was developed using the Microsoft Kinect platform with a focus on performance rather than precision, since the control loop design of the system enables efficient error correction. Finally, the tests performed on the system showed great promise, as the intuitive control mechanism made it easy to manipulate objects without much prior knowledge of the system.

A lot of improvement can still be made going towards a ubiquitous visual teleoperation system. Improving hand detection on low-priced hardware is inevitable, since it is important to correctly capture the users commands. With more advanced tracking systems for human joint positions, a general purpose algorithm transforming human to robotic arm joint positions could be developed, thus improving movement mimicking capabilities.

References

[1] P. K. Artemiadis and K. J. Kyriakopoulos, "Emg-based control of a robot arm using low-dimensional embeddings," *Robotics, IEEE Transactions on*, vol. 26, no. 2, pp. 393–398, 2010.

[2] J. Leitner, M. Luciw, A. Förster, and J. Schmidhuber, "Teleoperation of a 7 dof humanoid robot arm using human arm accelerations and emg signals," in *12th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Montreal, Canada*, 2014.

[3] G. S. Gupta, S. C. Mukhopadhyay, C. H. Messom, and S. N. Demidenko, "Master–slave control of a teleoperated anthropomorphic robotic arm with gripping force sensing," *Instrumentation and Measurement, IEEE Transactions on*, vol. 55, no. 6, pp. 2136–2145, 2006.

[4] B. Lim, S. Ra, and F. C. Park, "Movement primitives, principal component analysis, and the efficient generation of natural motions," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 4630–4635.

[5] B. C. Kuo, *Automatic control systems*. Prentice Hall PTR, 1981.

[6] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[7] M. de La Gorce, D. J. Fleet, and N. Paragios, "Model-based 3d hand pose estimation from monocular video," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 9, pp. 1793–1805, 2011.

[8] B. Stenger, P. R. Mendonça, and R. Cipolla, "Model-based 3d tracking of an articulated hand," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 2. IEEE, 2001, pp. II–310.

[9] J. M. Rehg and T. Kanade, "Visual tracking of high dof articulated structures: an application to human hand tracking," in *Computer VisionECCV'94*. Springer, 1994, pp. 35–46.

[10] C. Qian, X. Sun, Y. Wei, X. Tang, and J. Sun, "Realtime and robust hand tracking from depth," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014, pp. 1106–1113.

[11] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.

[12] J. Tompson, M. Stein, Y. Lecun, and K. Perlin, "Real-time continuous pose recovery of human hands using convolutional networks," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 5, p. 169, 2014.

[13] M. Šarić, "Libhand: A library for hand articulation," http://www.libhand.org/, 2011, version 0.9.

[14] R. Smits, "Kdl: Kinematics and dynamics library (2001)," 2013.