Collision Monitoring for a Mobile Manipulator Based on Biologically-Inspired Dynamical Systems for Movement Generation

Sreenivasa Hikkal Venugopala, Zain Ul Haq, and Urvashi Negi
Advisor: Djordje Vukcevic

***Abstract*— The computations involving the robotic manipulators, that is robotic arms and the mobile manipulators, that is mobile base (robot base) up until recent times are limited in the degree of operation and have several restrictions due to rigid movements, the control algorithms for these manipulators are usually complex involving various operations such as planning, perception, and so on. This makes the computations complex and time consuming and make it hard for its usage in the dynamic environments involving other robots, humans. To overcome these issues, in this research work, we propose to extend the implementation of the control algorithms which is implemented based on collision monitoring [1] and collision avoidance [2] for controlling the movements of the robotic arms from colliding with each other and to extend it to avoid collision with the mobile base as well. Along with this, we also implement the control algorithms for mobile base that helps in monitoring and avoiding the collision with any obstacles and helps in safe maneuvering towards the goal.**

I. Introduction

The robotic arms or the robotic manipulators are used in various applications ranging from industrial applications to domestic applications. They are widely used in industrial applications such as to support human in the production lines of heavy machineries for the purpose of assembling spare parts, welding, painting and so on, and in domestic applications such as robotic prosthetics, robotic assistants, and so on. Along with these applications, A robot as a whole, that is including both arms and the mobile base, poses useful in various applications such as warehouse management. All these applications require more robust and highly adaptable software and hardware components. For increasing the efficiency and the robustness of the functioning of the robotic manipulators in industrial applications, a precomputed trajectories will be provided [2], but these precomputed trajectories will not consider the dynamic movement of human and other obstacles. On the other hand, various operators are used to control the movement and functioning of the robotic prosthetics, but this does not provide free flow motion for the arms or prosthetics. On contrary it takes a lot of time and effort to train the robotic manipulators to perform a safe and free flow movement, but this is a high cost solution which is not weighable in all scenarios.

One of the solution approach to handle the control algorithm problem is to make use of the dynamic movement primitives, these are generated during the movement of the robot manipulators resulting in real-time collision monitoring and avoidance. There were many approaches proposed as a solution to this problem, based on the approaches proposed by [2] and [1] a library named 'Implementation of biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance [10]' was developed (`https://github.com/HBRS-SDP/sdp_ss20_collision_monitoring_for_robotic_manipulators`). This library handles the collision monitoring and collision avoidance for robotic arms in real-time and it incorporates the idea of dynamic movement primitives and the potential field method for active collision monitoring and avoidance. Here the dynamic primitives and the obstacles are considered as the basic primitives having shape and volume. The obstacle is avoided by planning a trajectory in such a way that a steering angle is computed in a potential field which steers the arms away from the obstacle and then converge towards the goal.

In this work, we propose to implement and extend the previous work by formulating the mobile manipulator or the robot base as a 3D box shape and extend the control algorithm to avoid and monitor the collision between the robot arms and its base. Along with this we also implement the control algorithm for the base to avoid and monitor the base from colliding with other obstacles which can be transformed into one of the shapes: sphere, cylinder, capsule or a box. Initially we planned and tried to model the base as a convex hull using the approach proposed by [8], and it turned out to be complex and cost ineffective for this application. To overcome this, we modeled the robot base to be a 3D box and calculate the distance between other dynamic primitives which are discussed in the future sections. Extension of previous work to monitor and avoid collision between the arms and the base, along with implementation of the control algorithm for controlling the base from avoiding and active monitoring of collision makes the robot more robust, adaptable, and safe to use in dynamic environments including for domestic purposes.

This report is further structured as follows. Section 2 describes modeling and calculating the distances between the dynamic primitives, Section 3 highlights the implementation details, Section 4 describes various experiments conducted, and Section 5 provides information on use cases followed by the conclusion.

II. Approach

A. Distance between primitives

In previous work, the links and joints, along with the end-effector and obstacles were modeled as spheres and cylinders. In this work, along with the previous implementations, we model the robot base as a 3D box, this helps in easier

calculations for the distance between the arms and robot body, and robot body and other obstacles. Following we will discuss the distance calculations between various primitive objects.

1) Distance between two spheres: We calculate distance between two spheres using below formula:

$$Distance = |\bar{C_1} - \bar{C_2}| - (r_1 + r_2) \tag{1}$$

where $\bar{C_1}$ and $\bar{C_2}$ represent the center points of the two spheres, and r_1 and r_2 are the radii of the two spheres and the same is visualized in Figure 1.

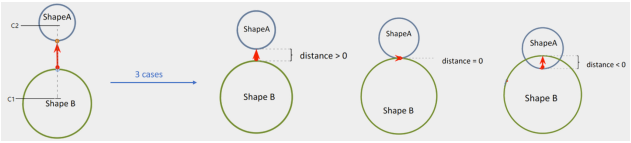

Fig. 1: Calculation of distance between two spheres

2) Distance between sphere and cylinder: We calculate the distance between the sphere and a cylinder using below equation and the same is shown in the Figure 2.

$$Closest point = X = \begin{cases} x = x_1 + \lambda(x_2 - x_1) \\ y = y_1 + \lambda(y_2 - y_1) \\ z = z_1 + \lambda(z_2 - z_1) \end{cases} \tag{2}$$

$$\lambda = \frac{(c - m_1).(m_2 - m_1)}{La^2} \tag{3}$$

$$Distance = |X - C| - (r_a + r_s) \tag{4}$$

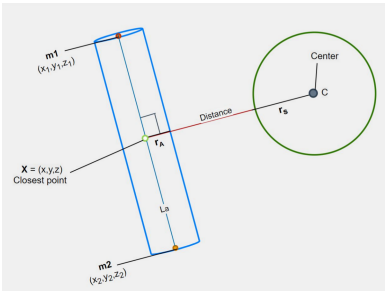

Fig. 2: Calculation of distance between a sphere and a cylinder

3) Distance between two cylinders: We compute the distance between the two cylinders based on the following four cases:

1) m1 and m2 can be perpendicularly projected on to La.
2) m1 and m2 cannot be perpendicularly projected on to La.
3) Only m2 can be perpendicularly projected on to La.
4) Only m1 can be perpendicularly projected on to La.

Here, m1 and m2 refers to the starting point and end point of the symmetrical axis of shorter cylinder, La is the symmetrical axis of the longer cylinder. This is shown in Figure 3.

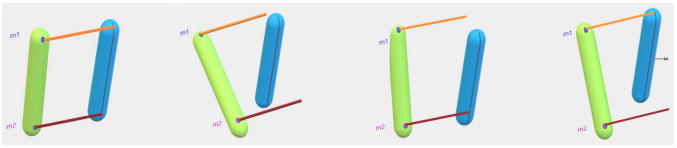

Fig. 3: Calculation of distance between two cylinders

4) Box configuration and closest point on the box: As an extension to the previous work, here we have modeled the robot base as a 3D box or of cuboid shape. This box is considered as Axis Aligned Bounding Box (AABB) [9], it means that the AABB doesn't have to have equal length, width, and height, but one condition here is that once it is configured the box should not be rotated. These AABBs are pivotal towards spatial partitioning. An example visualization is provided in Figure 4. These boxes can be stored in two ways, first is to store the leftmost and rightmost corners and the second way is to store the center point of the box and a vector representing how far it extends in each x, y, z directions.

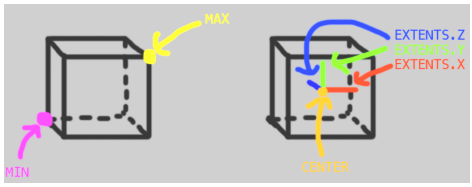

Fig. 4: Representation of AABB, image referenced from [9]

Initially we check if the given point is inside or on the box by comparing it to the position of leftmost corner (min) and rightmost corner (max). Since box has many edges and sides, we do have to figure out the closest point on the box that is closest to the obstacle. The algorithm to determine the closest point shows that the point will be clamped to AABB, that is we check if the point is outside the box and then compare the position of point with respect to min and max of the AABB and clamp the point to the nearest edge based on this comparison. If the point is inside the box, then the algorithm just returns the position of the point itself.

5) Distance between box and sphere: To find the distance between the box and the sphere, first we have to consider the distance between the center of the sphere and the closest point on AABB. To find out the closest point of AABB we consider the sphere center to be the point of consideration. Once we have the closest point on box, we calculate the distance by using Equation 5. If this distance is less than the radius of the sphere, then we say that the box and sphere are intersecting [9].

$$Distance = Sphere.Position - ClosestPoint \tag{5}$$

6) Distance between box and cylinder: A most widely implemented algorithm for collision detection in most physics engine used for game developments is GJK [8], this algorithm is based on creating convex hulls from two colliding convex geometry shapes by using minkowski sums [11] that

is by finding all difference of all the points on one object surface to the other.

We have minimized the implementation for our use case with two different complexities, one by considering only all the corner points and face centers of the box and the other by using all the edges of the box to calculate distance between the box and the other objects to avoid collision. This is visualized in Figure 6.

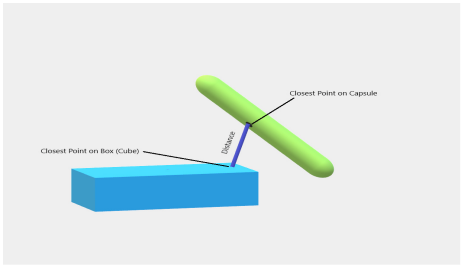

Fig. 5: Calculation of distance between a box and a cylinder

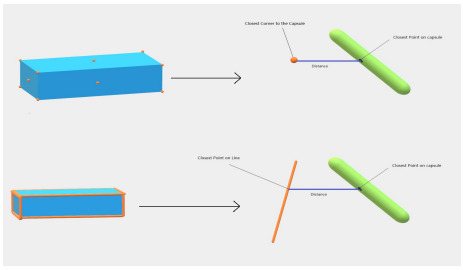

Fig. 6: Determining the closest point on the AABB.

To find the distance between the box and the cylinder, fist we have to determine the closest point on the cylinder or capsule using the Equation 2. We then clamp this closest point on to the AABB to find the closest point on the box. Once we have the two closest points, we consider the closest point of the cylinder or capsule to be the center point of the sphere and continue with the calculations by using the Equation 5. If this distance is less than the radius of the sphere (that is considered from the closest point on the cylinder or capsule to the edge of cylinder or capsule), then we say that the box is colliding with the cylinder or sphere [9]. This is visualized in Figure 5.

B. Potential field calculations

In previous work, to avoid and monitor the collision between the robotic arms and the obstacles, the algorithm from [2] is implemented. This algorithm is based on the dynamic movement primitives (DMP) and is used to generate the new trajectories and velocities in real-time as given in below equations,

$$\dot{v} = K(g-x) - Dv - K(g-x_0)s + Kf(s) + p(x,v) \tag{6}$$

Where g is goal position, x is current position, x_0 is starting position, v is current velocity, s is phase variable, D is damping constant, and K is spring constant. The term $p(x,v)$ refers to the potential field generated by the obstacles which in turn helps in avoiding obstacles.

This potential field is calculated by determining the steering angle (Figure 7) between the velocity vector and the vector from the obstacle to the end effector as shown in 7. This steering angle provides information on the sharpness of the steering of end effector from obstacle. Using this angle, the potential field is calculated as given in Equation 8.

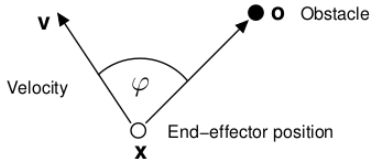

Fig. 7: Graphical representation of steering angle [2]

$$\varphi = cos^{-1}\left(\frac{(o-v)^T v}{\|o-x\| \cdot \|v\|}\right) \tag{7}$$

$$p(x,v) = \gamma \sum_i R_i v \varphi_i exp(-\beta \varphi_i) \tag{8}$$

Where γ and β are tune-able constants, and R is a rotation matrix which rotates by the axis $r = (o-x) \times v$ with an angle of rotation of $\pi/2$. This rotation matrix can be calculated using the following approach described in [6]. The final potential field is the sum of the potential fields generated by all the obstacles. The final equation converges towards the goal avoiding the obstacles as given below.

$$\dot{v} = K(g-x) - Dv + p(x,v) \tag{9}$$

C. Collision monitoring library

This work is an extension of the the previous work. Here along with the robot arms from previous implementation, we also model the robot base such that the arms do not collide with the base while performing some actions. Along with this extension, we also implemented the base control algorithm which actively helps in monitoring and avoiding collision between the robot base and other obstacles in the workspace.

In order to monitor the distance between the links of the robot arm, obstacles, robot base in the workspace, the links are modeled as the cylinders, end effectors are modeled as spheres, and the robot base is modeled as the 3D box. This type of modeling helps in easier calculations of distance between two primitives which results in monitoring of the self collision of the arms and base. This kind of modeling helps in easier calculation of potential fields, determining the nearest points or closest points on the primitives and obstacles that helps in calculating the distances, which in turn results in real-time collision monitoring and avoidance.

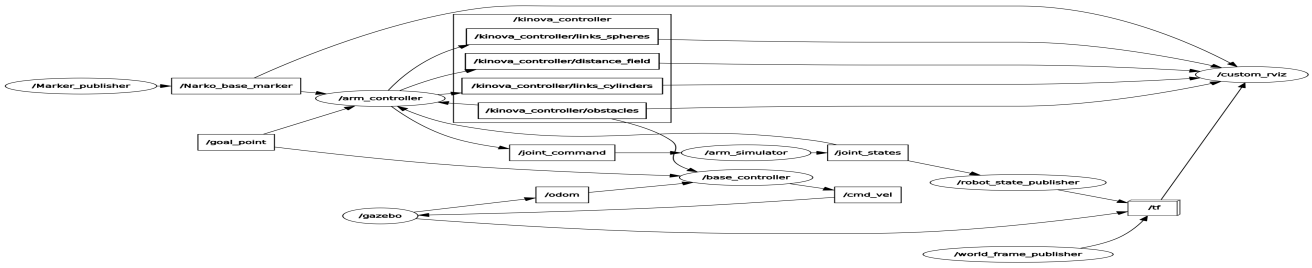

Fig. 8: ROS node graph showing the communication flow between the subcomponents of the package.

D. Controller

In this work, we follow the same architecture for the base controller as the previous work. Since we perform the collision monitoring and avoidance between the arms and the base, both the arm controller and the base controller are necessary. As per the previous work, the arm controller is controlled via a control loop which executes at a specific rate and provides the geometric locations which updates the velocity of the arms in real-time. Similarly, the base controller is also controlled via a control loop which executes at a specific rate and provides information on the geometric location of the mobile base and the velocity updates. This real-time information on the positions of arm and the base helps in quicker calculation of distances and helps in collision avoidance and monitoring with self and other obstacles in the workspace.

III. Implementation

In this work, we built the extension and the base controller implementation on the ROS platform, and tested with ROS kinetic and melodic versions. There are various libraries such as Kinova Kortex library [7], this library provides various high and low level control APIs for handling the robotic manipulators, Kinova_arm package that provides the design details, and implementation details for the robotic arms. Along with these libraries, in this work we make use of narko_description for creating the robot mobile base and connect the arms and the base. The final standalone library does not depend on any ROS components, and these ROS components are used only for testing and simulation of the collision avoidance and monitoring library.

The standalone library for collision monitoring and avoidance is purely developed using the C++ programming language, we make use of ROS and python for testing, simulation, and publishing of the robot base nodes. Similar to the previous work, the package is is divided into three subcomponents, the controller for base and arms, simulator and visualization. From previous work we have the arm controller implementation based on [2], and as an addition to this, we extended by adding the robot base as a box primitive to monitor and avoid the collision of arms with robot base. Addition to this, we implemented a base controller that helps in avoiding and monitoring the collisions with obstacles in the workspace. To visualize this, we have implemented a differential drive setup for movement and visualization of robot.

Similar to previous implementation, here we have implemented the collision avoidance part of the library in a control loop, that executes continuously to obtain the shortest distance vectors from the robot base and the obstacles. This shortest distance is then used for calculating the potential field along with the current velocity resulting in a new velocity and path for avoiding the collision and maneuvering towards goal position. This new velocity is then provided to the NarkinBase class for mapping the new velocity from Cartesian space to the robot base space.

The simulation software takes the new velocity, obstacle position and information on the potential field and generates a new position for the robot, and this new position is then passed on to the base controller such that it avoids the collision with obstacle and moves towards the goal position. This works in a loop until the robot reaches the goal position.

A basic ROS node graph showing the communication flow between the subcomponents of the standalone package is shown in Figure 8. Here we use the interfaces to add the obstacle to the workspace, and also to provide the goal position. For visualization, we make use of the in-built robot_state_publisher node and Rviz.

IV. Evaluation

In this section, we will discuss about experimental setup and evaluation of the implemented library.

A. Collision Monitoring

As an extension of the previous work, the computations behind the control algorithm is purely based on mathematics and hence the library do not have any practical measure of accuracy. To verify the computations and working of the library, we make use of ROS for creating an working environment which helps in simulation of working robot which is controlled using the implemented control algorithm. To have the consistency and the similar speed compared to previous work, this extension work is also implemented using C++. The performance test of this library was performed on Ubuntu 16.04 and Ubuntu 18.04 machines, including Ubuntu virtual machine.

For evaluation of this work, we make use of ROS simulations, where the kinova interfacer provides an interface for interacting with the robot workspace for adding the

obstacles, and providing goal points. Here we evaluate the working of the collision monitoring and avoidance between the robot arm and the robot base, along with this we also evaluate the collision avoidance between the obstacles in the workspace and the robot base. To achieve the similar speed as previous work, for testing purpose we have made use of the same values for k, d, gamma, and delta that are used in the calculations of new velocity, potential field, and steering angle.

For the visualization of the robot base we have used narko description which provides the robot description in the for of urdf fines and meshes, and we have used kinova_arm package which provides the robot description of arm in the form of urdf file for arm visualization. Each link in the arm is modeled as capsules or cylinders and the robot base as 3D box or cuboid. The same is visualized in Figure 9.

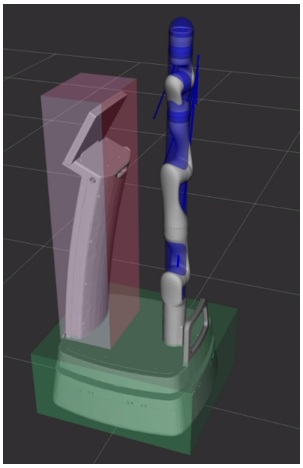

Fig. 9: Visualization of robot along with the modeled primitives.

1) Collision avoidance between arm and robot body: In this work, to avoid the self collision between the arm and the body, we have modeled the robot base as to be an obstacle with respect to the arm controller. The control algorithm treats this cuboid or the robot base as an obstacle and when ever the arm moves near to the body, the control algorithm generates the signal for active monitoring to avoid the collision. This is shown in Figure 10.

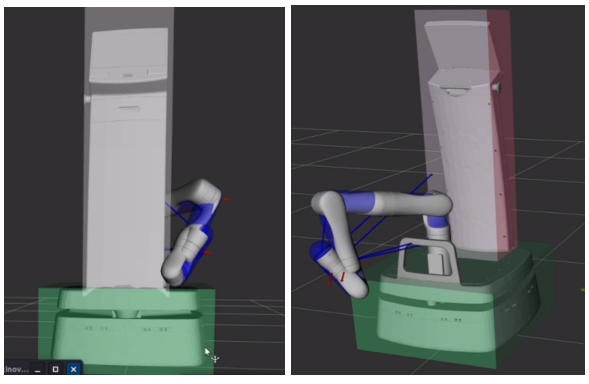

Fig. 10: Active monitoring of self collision.

The control algorithm actively monitors the movement of the robot arm even when they are stand still. As seen in the left image of the Figure 10, when the robot arm moves near to the robot body, the control algorithm generates signals by making use of the potential field and the new velocities to avoid colliding with the body, this is shown using the red arrow in the simulation. Similarly, when the robot arm moves near the robot base, the red arrows from the active monitoring are visible in the simulation.

2) Collision avoidance between robot and the obstacles: Along with the monitoring of the self collision, we have also implemented the base control algorithm that actively monitors and avoids collision with obstacles in the workspace. For simulation of the robot movement, we have implemented a two wheel differential drive robot using the narko description and for movement of the robot we have implemented a PID controller. Here we have implemented the algorithm for calculation of the distance between the base and the obstacles. The initial distance between the obstacle and the robot base is visualized in the Figure 11.

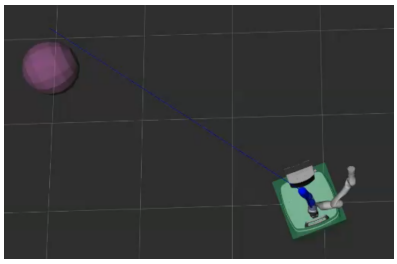

Fig. 11: Visualization of initial distance between robot and obstacle.

During testing of the base controller algorithm for avoiding collision with obstacles, we noticed that due to some calculation errors with respect to the translation of obstacle pose into robot base space, the algorithm for collision avoidance does not work properly and proper conversion mechanism and optimization techniques needs to be implemented to solve the problem. The result of this calculation glitches is visualized in Figure 12. Currently, the object pose is translated on to the arm space and the initial position of the obstacle varies as and when the robot moves. Due to time constraints, this was not corrected in this implementation.

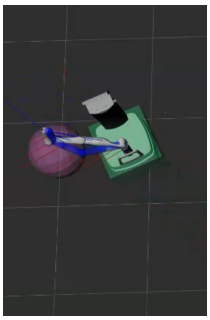

Fig. 12: Visualization of wrong movement generation due to calculation errors.

V. Use cases

A. Arm

Since this work is an extension of previous work, all the features that were implemented related to the arms still functions and can be used to create and experiment with multiple number of arms. Considering single arm, the arm controller looks and helps in avoiding and monitoring the collision of end-effector with the obstacles in the workspace. Considering dual arms, the links and joints of the arms are represented as obstacles, hence both arms do not collide with each other and also they do not collide with the obstacles. Any number of obstacles can be added to the workspace using the interfacer provided in the library. Similarly, the library is capable of handling more than two arms and they have to be configured before execution of the library.

B. Base

In this work, the robot base is modeled as a 3D box or can be referred to as a cuboid. This cuboid is treated as an obstacle from the arm controller point of view and this helps in avoiding and monitoring of self collision between arm and the base. This cuboid representation acts as a generic representation since most of the robot's base is either of a cuboid shape or of cylindrical shape and can be enclosed using a cuboid. Along with this, the base controller implementation (currently buggy) looks out for obstacles in the workspace and actively monitors and avoids collision.

VI. Conclusion

This work is an extension of [10], where the existing library is extended to perform collision monitoring and avoidance of robot arms with robot base. To achieve this, we explored various modeling techniques and algorithms of which [8] provides promising performance with respect to distance calculation between objects, but implementation and testing of this algorithm requires more complex computations and hence we chose to model the robot base as Axis Aligned Bounding Boxes (AABB) [9] which are easier to store and computations are simpler as discussed in the previous sections. This AABB modeling approach also provides easier and simpler ways for performing unit tests.

As an extension to the existing library, we modeled the robot base as an obstacle with respect to the arm controller such that it helps in avoiding and monitoring self collision. To this extent we were successful in achieving the active monitoring of the self collision. Although this implementation works as expected, there is always a room for optimization.

Along with this we also implemented a base controller which actively monitors the collision between the obstacles and the robot base. We implemented a PID controller and two wheel differential drive robot using the narko description for movement control of the robot. We were successful in calculating the distance between the obstacles and the robot base. But due to error in calculating the translation of new positions and velocities from one space to other, the active monitoring does not work as expected and due to time constraints, we were not able to fix it.

As future work, we would like to stress more on the optimization part of the self collision and to implement more effective control algorithm for the robot base.

Appendix

The code for the collision monitoring library is accessible at the following link: `https://github.com/Sreeni1204/sdp_ws20_collision_monitoring_for_mobile_manipulators`

Acknowledgements

We would like to thank Djordje Vukcevic for his support and motivation throughout the course of this project.

References

[1] O. Khatib, Real-Time Obstacle Avoidance for Manipulators and Mobile Robots The International Journal of Robotics Research, vol. 5, no. 1, pp. 9098, Mar. 1986, doi: 10.1177/027836498600500106.

[2] H. Hoffmann, P. Pastor, D.-H. Park, and S. Schaal, Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance, in 2009 IEEE International Conference on Robotics and Automation, Kobe, May 2009, pp. 25872592, doi: 10.1109/ROBOT.2009.5152423.

[3] M. S. Fadali and A. Visioli, Introduction to Digital Control, in Digital Control Engineering, Elsevier, 2013, pp. 18.

[4] S. Adee, Dean Kamens Luke Arm Prosthesis Readies for Clinical Trials - IEEE Spectrum, IEEE Spectrum: Technology, Engineering, and Science News, Feb. 01, 2008. https://spectrum.ieee.org/biomedical/bionics/dean-kamens-luke-arm-prosthesis-readies-for-clinical-trials (accessed Jun. 25, 2020).

[5] F. Janabi-Sharifi and D. Vinke, Integration of the artificial potential field approach with simulated annealing for robot path planning, in Proceedings of 8th IEEE International Symposium on Intelligent Control, Aug. 1993, pp. 536541, doi: 10.1109/ISIC.1993.397640.

[6] Belongie, Serge. "Rodrigues' Rotation Formula." From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein. https://mathworld.wolfram.com/RodriguesRotationFormula.html

[7] Kinovarobotics/ros_kortex. Kinova Robotics, 2020.

[8] E. G. Gilbert, D. W. Johnson and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," in IEEE Journal on Robotics and Automation, vol. 4, no. 2, pp. 193-203, April 1988, doi: 10.1109/56.2083.

[9] AABB 3DCollisions - `https://gdbooks.gitbooks.io/3dcollisions/content/Chapter1/aabb.html`, Accessed on 03/07/2021

[10] A. Gomez, S. Parra and B. Penfold, Implementation of biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance, 2020.

[11] Coulombe, A., and Lin, H.-C. (2020). High Precision Real Time Collision Detection. ArXiv Preprint ArXiv:2007.12045.