UDC 62-526
Kutovyi Oleksandr, Seminska Natalia
National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”, Kyiv, Ukraine

Optimizing hyperparameters for mobile robot platform control

Introduction. In recent years industrial manipulators have become the most desired feature in factories built by modern standards due to precision at all times, high repeatability of processes and versatility of usage. It is a widely argued question whether robots will replace human workers, however, it is sure that such mechanisms will be in a high demand fort the nearest decades.

In todays reality manipulators are mainly applicable to large-scale factories because of reasons mentioned above, nevertheless small businesses are also “eligible” to purchase one due to the variety of models available on the market.

With that concern author has chosen a mobile platform based on 4 mecanum wheels and 5R serial manipulator on top, namely Kuka YouBot. This is a redundant robot, meaning it has more than 6 degrees of freedom, it can move in a tight work-space environment and perform simple tasks with it`s default gripper.

In recent years, this platform has been chosen to accompany world series mechatronics competition. User may use the default Kuka`s software as well as write it`s very own to perform more complex task e.g. camera tracking or GPS-based odometry.

Pic.1 Kuka YouBot @ https://www.kuka.com/

Main part. The purpose of this work is to combine knowledge about different types of trajectory planners and controllers and suggest best combination of both.

Trajectory planners act mainly by taking as an input start position and desired position, thus yielding an interpolated path between them. It worth noting, that if user want to interpolate path with n-degree polynomial it has to be two more than the derivative of position e.g. if we want to take into account acceleration – 3rd derivative, we have to use 5th degree polynomial, so it would be twice differentiable.

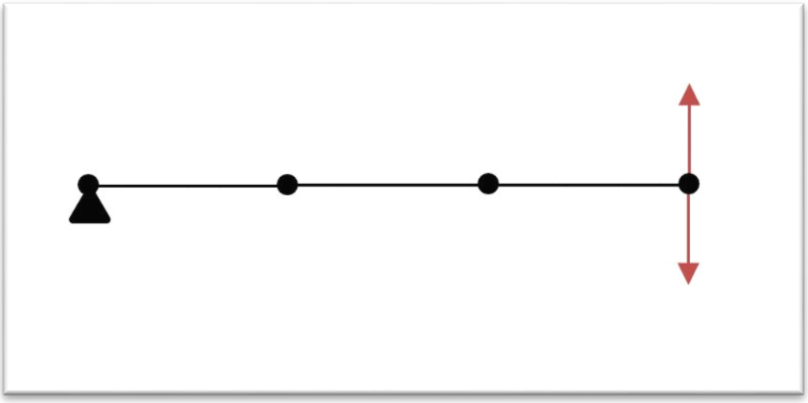
The higher the degree then the transition is smoother, but more computations required. The common practice is a 5th degree polynomial, which results in smooth motions and computationally efficient to use in real-time applications.

It is reasonable to specify the trajectory in in more strict form, rather than just start and goal configurations. And there are 3 main methods for this: Cartesian trajectory, Screw trajectory and trajectory through via-points.

It is worth noting that in 3-D space end-effector position is described as a simultaneous rotation and translation relative to some fixed coordinates. What a Cartesian trajectory planner does is translating the manipulator from initial to goal configuration along the strait line and at the same time rotating it. A Screw trajectory planner, on the other hand, treats rotation and translation differently. Any movement in spatial space can be described as a twist and translation along some screw axis with a designated angular and linear speeds. It is convenient to represent it in a matrix form with angular and linear components. Then Screw trajectory is as it sounds. In practice it sometimes may be an arc or helix depending on inputs.

The via-points trajectory planner is used, when user specify only points that manipulator has to follow and speeds, disregarding the trajectory between them.

As a result of simulations in CoppeliaSim environment some line can be drawn. In most cases it is a good decision to use a Screw trajectory, because it is smooth and works quite well with self-intersection avoidance algorithms. However, when robot begins or ends its motion in a singularity position – a position, where movement cannot be made in any arbitrary direction, e.g. 3R robot with all angles set to 0 deg. thus it cannot move to the origin instantly; this method generates points outside the feasible working space.



Pic.2 Robot at a singularity

In such cases a Cartesian trajectory is preferred. It also takes shorter time to follow because straight line is ultimately the shortest path between two points. Yet, this method does have constraints due to self-intersection. And because of this fact additional joint movements need to be introduced.

The author has implemented a simple method, so when a joint angle reaches its bound position a corresponding to that joint column in a Jacobian matrix is all set to zero, thus any further motion of that coupling does not affect the end-effector position.

The via-point method is best suited for application where manipulator has to avoid obstacles. A set of points in space can be sampled randomly and then creating a links between them if only those lines do not intersect objects. There are many ways how to define an object in space and check for intersection, but there two ends of spectrum. The computationally efficient, yet not very precise is a way of describing an obstacle as set of connected spheres and subsequently checking the distance between their centers and manipulator links and joints. On the other hand, there is a way of describing the robot itself and obstacles as point clouds, creating oriented bounding boxes or even complete mesh geometry and after that check for the amount of intersection between them. Due to the progress in computer hardware and rapid increase in computational power latter method is becoming an industry standard but comes at a high price with respect to money and time investments.

Robot controllers are an essential part of robust and reliable algorithm. They can ease life of the user by taking many parts of computation from overall process. Some of them support only second order dynamics, meaning that they take velocities as an input; others may rely on third order dynamics, thus taking forces and accelerations. Latter variant is more suitable for the real-life applications because it considers

gravitational forces and inertia properties of manipulator. Such controller is essential when user wants to specify and maintain forces of some magnitude and direction.

The author has written a software analog to a second order hardware controller and run multiple simulations. There three different options: feedforward only control, feedforward-plus-P control and feedforward-plus-PI control. The results are given in Pic.3 and performance is measured based on end-effector error twist at a given moment of time.

Here is overall formula for the controller:

$$V(t) = \left[Ad_{X^{-1}X_d}\right]V_d(t) + K_p X_{err}(t) + K_i \int_0^t X_{err}(t)dt$$

Where:

V(t) - the commanded end-effector twist expressed in the end-effector frame {e};

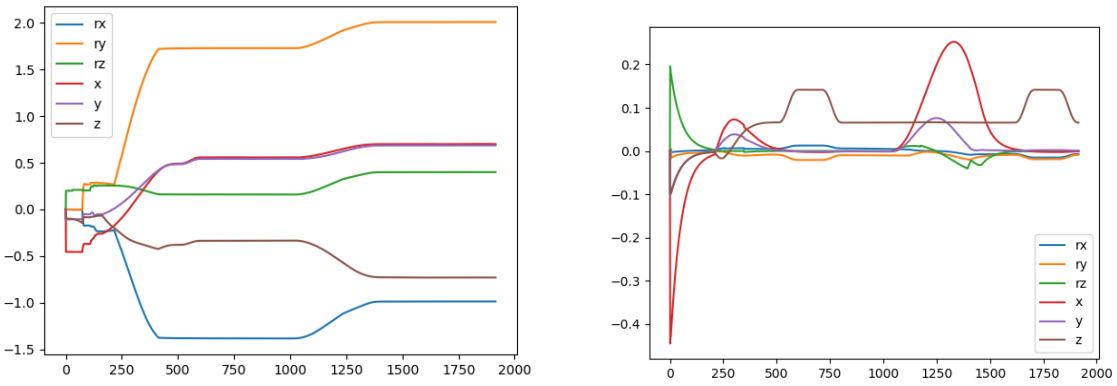
X – The current actual end-effector configuration;

X_d - The current end-effector reference configuration;

K_p and K_i - The PI gain matrices;

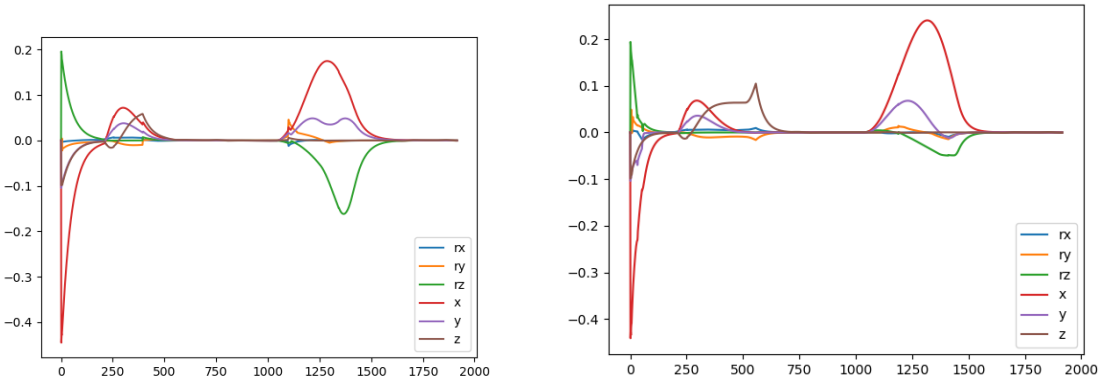
V_d – feedforward reference twist;

X_{err} - The error twist.



1 2

Pic. 3 The plots of error twist Xerr: 1 – feedforward only, 2 – feedforward plus P, 3 - feedforward plus PI fine-tuned, 4 - feedforward plus PI. X-axis – time in ms, Y-axis error in meters



3 4

Pic. 3 (continue)

It is worth noting, that manipulator begins its motion with an initial error, so we can see it at an initial moment of time. This initial error shows us the difference between behaviors under different type of controllers. As can be seen, feedforward only commands preprogrammed path and does not take feedback into account. P type controller is good at reducing initial error, but further distortions cause instability and movement outside the planned trajectory. PI controller outperforms them all, but requires accurate Kp and Ki, otherwise may behave similarly to P controller.

It is worth noting that there is a plenty of room for fine-tuning and modifications to the algorithm. Beginning with third order controller and up to better self-collision and obstacle avoidance algorithm. For the complete integration to the human work environment it is crucial to implement CCTV odometry and more importantly neural networks for obstacle recognition. Recent studies show promising results for middle-grade hardware and live-streaming of data to neural networks without any losses in conclusion accuracy.

Conclusions:

1. A set of experiments were conducted in order to obtain algorithm best suited for Kuka YouBot control in the designated environment.
2. A theoretical basis for upper conclusion was given.
3. Further enhancements in both self-collision and obstacle avoidance algorithms will be made.

References:

[1] Kevin M. Lynch and Frank C. Park 30.11.19 “MODERN ROBOTICS MECHANICS, PLANNING, AND CONTROL”

[2] Anthony A. Maciejewski and Charles A. Klein The International Journal of Robotics Research, 1985, “Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environment” DOI: 10.1177/027836498500400308.

[3] Tuomo Kivel`a, Jouni Mattila, Jussi Puura, and Sirpa Launis 2017 “Redundant Robotic Manipulator Path Planning for Real-Time Obstacle and Self-Collision Avoidance” conference paper version for RAAD 2017 conference. DOI: 10.1007/978-3-319-61276-8_24

[4] Agostino De Santis, Alin Albu–Sch¨affer, Christian Ott, Bruno Siciliano, and Gerd Hirzinge 2014 “The skeleton algorithm for self-collision avoidance of a humanoid manipulator” DOI: 10.1109/AIM.2007.4412606

[5] https://www.coppeliarobotics.com/

[6] https://www.kuka.com/