Non-Prehensile Manipulation By Wheeled Mobile Robot

[1]Shayak Bhadraray, [2]Narayanan Ramasamy, [3]Haard Shah, [4]Surabhi Gade, [5]Neel Adke
[1,2,3,4,5] College Of Engineering, Northeastern University
[1]bhadraray.s@northeastern.edu, [2]ramasamy.n@northeastern.edu, [3]shah.haar@northeastern.edu, [4]gade.s@northeastern.edu, [5]adke.n@northeastern.edu

***Abstract*—**While manipulation of objects serves as one of the most essential applications in the field of robotics, one can argue upon the complications which one might face while considering a grasping-based approach for object movement. A simple pick and place manipulator can achieve desirable results, however the mass of the object to be moved, the shape of the object as well as the number of contact points between the manipulator and the object can further complicate the procedure. Hence, comes the importance of "Non - Prehensile Manipulation". The following report is a review of the various implementation algorithms for the same, applied on wheeled robots for the further research scope of optimizing the existing work**.**

Keywords—Non-Prehensile, Manipulation, Gazebo, Manipulator, Grasping, Mobile Robot

I.INTRODUCTION

One of the most basic tasks for a robotic manipulator is to move an object from one place to another. A common solution is to equip the manipulator with a gripper and adopt the pick-and-place approach. By designing the grasp to resist all forces that could reasonably act on the object during the motion, one might achieve desirable results. If the object is too large to be grasped or too heavy to be carried, however, this approach fails. In this report we examine pushing, which provides a simple and practical solution to the problem of positioning and orienting objects in the plane, particularly when the manipulator lacks the size, strength, or dexterity to grasp and lift them [1].

II.PROBLEM STATEMENT

This report aims at implementing / reviewing the techniques to manipulate an object with a nonholonomic mobile robot by pushing, which is a non-prehensile manipulation motion primitive. Such a primitive involves unilateral constraints associated with the friction between the robot and the manipulated object. Violating this constraint produces the slippage of the object during the manipulation, preventing the correct achievement of the task. The constraints ensure that the robot is constantly kept in contact with the object to be manipulated whereas a trajectory planning algorithm is implemented to drive the robot from its initial to final configuration. Hence, the object is manipulated to the desired goal and location as an optimized solution of both trajectory planning and contact slippage avoidance.

III. TERMINOLOGIES

Non-Holonomic Robot:

A non - holonomic robot is considered as a robot model which exhibits certain constraints when it comes to the aspect of the robot's mobility. Essentially in non-holonomic robots, the number of controllable degrees of freedom is less than the number of degrees of freedom which the robot experiences. As a result, we are bound to apply motion constraints called "Non - Holonomic Constraints". One such example of a robot is a differential drive robot. For the implementations further discussed in the report we have considered the Waffle model of TurtleBot3 as the target robot model (which is one of the popular available differential drive robots) for testing out the algorithms of our research.

Non-Prehensile Manipulation:

Manipulation by grasping is very restrictive. People manipulate objects in many interesting ways that do not involve grasping. Objects may be pushed, flipped, thrown, squeezed, twirled, smacked, blown, and so on. These are all examples of non-prehensile manipulation, which means manipulation without grasping [1].

ROS (Robot Operating System):

ROS or Robot Operating System is a collection of open-source software frameworks for robot software development. It is not an operating system but a modular framework that helps use custom services, nodes, and messages.

- ROS Nodes: ROS Nodes are individual nodes (or discrete programs) which compute desired data. Multiple nodes can be run simultaneously.
- ROS Topics: ROS Topics are data streams, i.e., they transmit data between nodes. A node can send data in a topic by publishing to it and can receive data from a topic by subscribing to it.
- ROS Messages: ROS Messages are messages that nodes send to each other via ROS Topics. These messages can consist of any number of individual pieces of data belonging to any data type.
- URDF: Also known as Universal Robot Description Format, is a file written in XML (Extensible Markup Language) used to describe the physical structure of various models.

- RQT Graph: It is a GUI Plugin used to visualize the ROS computation graph. It depicts the various nodes, the connections between them and the topics that each node sends to another.

Gazebo:
Gazebo is a 3D simulator for robot interactions. While ROS serves as the interface for the robot, combining both results in a powerful robot simulator. With Gazebo you can create a 3D scenario on your computer with robots, obstacles and many other objects, along with a powerful physics engine for simulating the effects of illumination, gravity, inertia, etc.

IV. METHODOLOGY

The following section of the report discusses the various aspects of the implementation of algorithms designed for mobile robots. "*Trajectory planning*" and "*Slippage avoidance constraint*" serve as the primary areas of the approach where one ensures the proper path is followed and the later complements it by ensuring the efficient contact of the robot with the object to be manipulated for the entire duration of the trajectory followed by the robot.

A. Trajectory Planning

A* Search Algorithm
The A* search algorithm efficiently finds a minimum-cost path on a graph when the cost of the path is simply the sum of the positive edge costs along the path. Given a graph described by a set of nodes N = {1, . . ., N}, where node 1 is the start node, and a set of edges E, the A* algorithm makes use of a heuristic cost to go function which estimates the actual cost-to-go to the goal from node [2].
In our case we consider the entire environment for the robot as a graph, where each node represents the configuration the robot can achieve in the simulation environment, Hence, a graph traversal approach, while considering all the static obstacles, is used to generate the optimal path to the goal. Following is a pseudo-code for implementing the same:

```
OPEN ← {1}
past_cost[1] ← 0, past_cost[node] ← infinity for node ∈ {2,...,N}
while OPEN is not empty do
  current ← first node in OPEN, remove from OPEN
  add current to CLOSED
  if current is in the goal set then
    return SUCCESS and the path to current
  end if
  for each nbr of current not in CLOSED do
    tentative_past_cost ← past_cost[current]+cost[current,nbr]
    if tentative_past_cost < past_cost[nbr] then
      past_cost[nbr] ← tentative_past_cost
      parent[nbr] ← current
      put (or move) nbr in sorted list OPEN according to
           est_total_cost[nbr] ← past_cost[nbr] +
                heuristic_cost_to_go(nbr)
    end if
  end for
end while
return FAILURE
```

Fig 1. Pseudo-code for A Search*

Drawbacks:

- This algorithm is complete if the branching factor is finite and every action has fixed cost.
- The performance of A* search is dependent on accuracy of heuristic algorithm used to compute the function 'H(n)'.
- The algorithm assumes a constant state of the map of the environment, which is not always the case in a practical implementation.
- The algorithm assumes all the configuration spaces are accessible at all times for the mobile robot, which is not always the case in a practical implementation.
- The algorithm brings no consideration of dynamic obstacles.

Rapidly Randomizing Trees (RRT)
A rapidly exploring random tree (RRT) is an algorithm designed to efficiently search nonconvex, high-dimensional spaces by randomly building a space filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem [3]. The key idea is to bias the exploration toward unexplored portions of the space by sampling points in the state space, and incrementally "pulling" the search tree toward them. Every time the robot is triggered to move to a specific goal configuration the navigation stack reconstructs a new tree with the expanding bias towards the unexplored regions near the goal configuration, once the tree is successfully constructed a search algorithm queries the nodes thus formed by sampling and thus provides the path for the robot to follow to reach the goal state. The pseudo code for the same is given as:

```
BUILD_RRT(x_init)
1  T.init(x_init);
2  for k = 1 to K do
3      x_rand ← RANDOM_STATE();
4      EXTEND(T, x_rand);
5  Return T
```

```
EXTEND(T, x)
1  x_near ← NEAREST_NEIGHBOR(x, T);
2  if NEW_STATE(x, x_near, x_new, u_new) then
3      T.add_vertex(x_new);
4      T.add_edge(x_near, x_new, u_new);
5      if x_new = x then
6          Return Reached;
7      else
8          Return Advanced;
9  Return Trapped;
```

Fig 2. Pseudo-code for RRT

Drawbacks:

- This algorithm is computationally extensive.
- One weakness of the basic RRT algorithm is that it does not take path cost into account.
- The navigation tree is basically reconstructed at every step when the navigation stack is called.

B. URDF Modelling

Upon further inspection of the present work being done in the field of "*non-prehensile manipulation by wheeled mobile robots*", it is observed that the number of points of contacts serve as an essential parameter to ensure minimum slippage between the robot and the object. Thus, the need to improvise upon the current model of TurtleBot3 Waffle arises and the need to perform URDF modelling to achieve the same is thus proposed.

URDF (Also known as Universal Robot Description Format), is a file written in XML (Extensible Markup Language) is used to describe the physical structure of various models. The default URDF files present in the open-source documentation of the TurtleBot3 waffle model loads the robot model as shown above in the introduction section of the report. Thus, we have edited the files for the description of the above referred differential drive robot to add two links to the either ends of the front of the robot, and create a flat plate shaped structure to thus keep in contact with the object to be manipulated in the open environment. The following code implementation shows how the same was done:

```
 <joint name="my_joint" type="revolute">
  <origin xyz="0.07 0.07 0.09" rpy="0 0 0"/>
  <parent link="base_link"/>
  <child link="board_link"/>
  <limit velocity="0" effort="0" lower = "0" upper = "0"/>
</joint>
<link name="board_link">
<visual>
  <geometry>
    <box size="0.01 0.15 0.15"/>
  </geometry>
  <origin rpy = "0 0 0" xyz = "0 0 0"/>
</visual>
<inertial>
  <mass value = "0.01"/>
  <origin xyz = "0 0 0"/>
  <inertia ixx="0.002" ixy="0.0" ixz="0.0" iyy="0.001" iyz="0.0" izz="0.0"/>
</inertial>
<contact>
  <lateral_friction value="1.0"/>
  <rolling_friction value="0.0"/>
  <stiffness value="1"/>
  <damping value="1"/>
</contact>
</link>
<gazebo reference = "my_joint">
  <provideFeedback>true</provideFeedback>
</gazebo>
<gazebo>
  <plugin name="ft_sensor" filename="libgazebo_ros_ft_sensor.so">
    <updateRate>20</updateRate>
    <topicName>gazebo_ft_sensor</topicName>
    <jointName>my_joint</jointName>
  </plugin>
</gazebo>
```

Fig 3. Edited URDF Code 1

Furthermore, as we realized the need to implement newer motion constraints for our further approaches, we came upon the need to install '*Force – Torque sensors*' on both of the links used to add the rectangular plate. Thus, creating a feedback system to read the amount of force being exhibited over the object being manipulated by the mobile base. The following shows the implementation for the same

```
<gazebo reference = "my_joint">
  <provideFeedback>true</provideFeedback>
</gazebo>
<gazebo>
  <plugin name="ft_sensor" filename="libgazebo_ros_ft_sensor.so">
    <updateRate>20</updateRate>
    <topicName>gazebo_ft_sensor</topicName>
    <jointName>my_joint</jointName>
  </plugin>
</gazebo>
```

Fig 4. Edited URDF Code 2

The result of the complete URDF model being imported in gazebo was as follows:

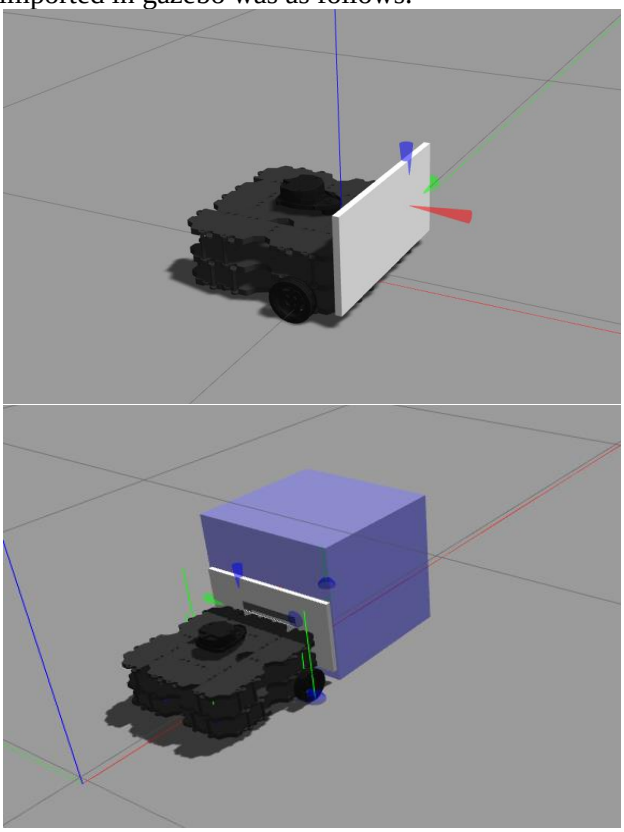

Fig 5. Robot visualized in Gazebo

C. Non-Prehensile Interaction/Mechanism

Virtual Dipole Fields:

Concept Overview:

Virtual potential field methods are inspired by potential energy fields in nature, such as gravitational and magnetic fields. From physics we know that a potential field P(q) defined over C induces a force F calculated as *'dP/dq'* that drives an object from high to low potential. In robot motion control, the goal configuration goal is assigned a low virtual potential and obstacles are assigned a high virtual potential. Applying a force to the robot proportional to the negative gradient of the virtual potential naturally pushes the robot toward the goal and away from the obstacles. The drawback of the basic method is that the robot can get stuck in local minima of the potential field, away from the goal, even when a feasible motion to the goal exists. In certain cases, it is possible to design the potential to guarantee that the only local minimum is at the goal, eliminating this problem.

Paper Implementation:

The specific paper consulted introduces a simple algorithm for non-prehensile object transportation by a pushing robot on a flat surface. The paper assumes that the global position and orientation of the robot and objects are known at all times. The system computes a dipole field around the object and moves the robot along the field. This simple algorithm resolves many subtle issues in implementing reliable pushing behaviors, such as collision avoidance, error recovery, and multi-robot coordination [4].

The system first defines a local coordinate frame whose origin is at the center of the object, with x-axis parallel to its desired direction of motion. The system then computes the polar coordinates (r, θ) of the robot in this local coordinate frame. The desired direction of motion of the robot in this local coordinate frame is given by (cos2θ, sin2θ). It is independent of the robot–object distance and the sizes of the robot and the object. The pseudocode of a similar implementation done by us is shown in the figure below:

```
compute(d = desired direction of motion,
            p = object position, q = robot position){
    x = d; y=rotate(x, π/2);
    θ = compute_angle(x, q-p);
    return  normalize(x cos2θ + y sin2θ);
}
```

Fig 6. Pseudocode for Virtual Dipole Fields

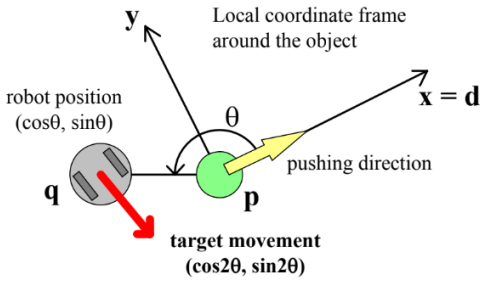

Fig 7. Scenario of Virtual Dipole Fields

Linear Time - Varying MPC:

Concept Overview:

Model Predictive Control (MPC) is a trajectory planning/tracking algorithm where the main idea behind the controller formulation is to optimize the future behavior of the robot across a finite prediction horizon over a finite number of steps. Hence, the conceptual overview for trajectory planning for the robot in an environment through an MPC controller formulation stands as follows: -

- The target position for the robot is specified to its controller.
- Intermediate steps are thus generated between the present orientation of the robot till the final desired orientation.
- Each step thus generated represents a future configuration for the robot which it needs to take in order to reach the target.
- Hence, the robot's behaviour is thus an optimized output w.r.t the next immediate step for the planned trajectory.
- The robot repeats the same procedure for a number of iterations, until the final position is reached.

Linear Time – Varying MPC based controllers provide us with the freedom to constrain linear combinations of plant input and output variables. Hence, allowing us to implicitly declare constraints over the kinematics of the robot. Hence, the specific paper referred here, uses LTV MPC to define the motion of the non – prehensile manipulation to be done while implementing an object slip avoidance constraint to ensure the object remains in contact with the robot while the process of manipulation is active.

Paper Implementation:

The specific paper consulted for replication of their results, proposes a technique to manipulate an object with a nonholonomic mobile robot by pushing, which is a non - prehensile manipulation motion primitive. Such a primitive involves unilateral constraints associated with the friction between the robot and the manipulated object. Violating the constraint produces the slippage of the object during the

manipulation, preventing the correct achievement of the task. A linear time-varying model predictive control was designed to include the unilateral constraint within the control action properly. Thus, producing the desired manipulation result.

Robot Modelling:

A detailed description of the modelling done for the interaction between the robot and the object, to define the physical parameters necessary for MPC and constraint formulation are provided in the paper referenced below [5]. 'Σw' and 'Σr' are defined as the global reference frame and the body frame attached to the center of the robot's axle, respectively. The Heading / Yaw angle of the robot is defined as 'θr'. The diagrammatic representation for the same is provided below.

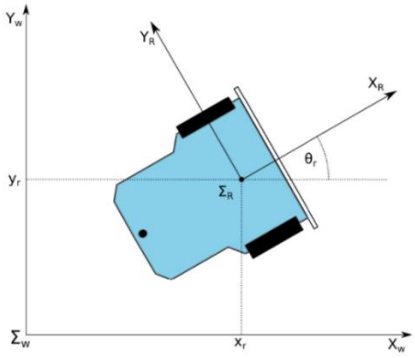

Fig 8. F.O.R for MPC Formulation

The length of the object (for this case, assumed as a box) is taken as '2s' and the interacting pushing forces on the either side of the robot – object interaction as shown below are assigned as f_{1R}, f_{2R}, f_{1L}, f_{2L}.

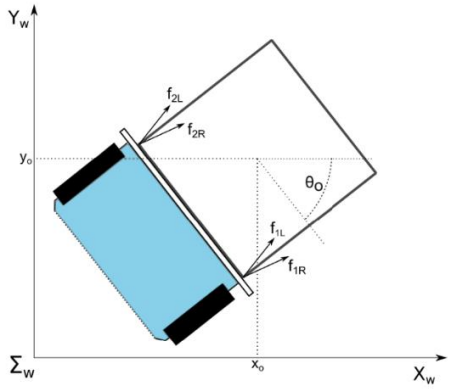

Fig 9. MPC Problem Formulation

Pushed Object Model:

A Matrix 'f_c' is defined by taking all the forces in consideration as: -

$$f_c = \begin{bmatrix} f_{1R} & f_{1L} & f_{2R} & f_{2L} \end{bmatrix}^T$$

Fig10. Matrix defining all forces in consideration

The Frictional Coefficient between the interacting surfaces is assumed to be known as 'μ', hence the friction cone angle between the robot – object surface interaction is denoted as: $\tan^{-1}(\mu)$. A matrix termed as 'Grasp Matrix (G)' is also defined to denote the interaction with the robot and the object termed as: -

$$G = \begin{bmatrix} \cos(\theta_r - \theta_\mu) & \sin(\theta_r - \theta_\mu) & s(\cos\theta_\mu + \sin\theta_\mu) \\ \cos(\theta_r + \theta_\mu) & \sin(\theta_r + \theta_\mu) & s(\cos\theta_\mu - \sin\theta_\mu) \\ \cos(\theta_r - \theta_\mu) & \sin(\theta_r - \theta_\mu) & s(\sin\theta_\mu - \cos\theta_\mu) \\ \cos(\theta_r + \theta_\mu) & \sin(\theta_r + \theta_\mu) & -s(\cos\theta_\mu + \sin\theta_\mu) \end{bmatrix}^T$$

Fig11. Grasping Matrix as defined by the implementation

Alongside to this, the paper further comments on the importance of '*Ellipsoidal Limit Surface'* and cites a paper which theoretically and experimentally shows that an ellipsoid fits to the maximum tangential force, in order to ensure maximum contact between two objects. Hence a matrix termed as 'H' is also defined which denotes the aspect of ellipsoidal approximation of the contact forces, in order to ensure minimum slippage between the object and the robot. The derivation for the same is discussed in the next section. Hence, the total external wrench '*w*' expressed whose torque is applied around the object's center of mass, exerted by the robot to the object can be described as: '$w = G\ f_c$'

Ellipsoidal approximation of the limit surface:

A limit surface is basically a matrix denoting the mapping between contact forces and sliding motions at a point of contact. The limit surface is constructed by computing the contact frictional forces and moment for each possible translational and rotational slippage of contact interface and also knowing the contact shape, contact pressure distribution, and friction coefficient between the bodies. Hence, $[f_x, f_y, m_z]^T$ denotes the limit surface for a point of contact where f_x is the force along x - axis, f_y is the force along y- axis and m_z is the moment along z – axis for the point of contact [6].

Hence to assume the normal reaction to be 'N' at the point of contact the model of an ellipsoidal approximation for the same, we apply the following constraints on the limit surface [7]:

$$\frac{f_x^2}{(f_t)^2_{\max}} + \frac{f_y^2}{(f_t)^2_{\max}} + \frac{m_z^2}{(m_z)^2_{\max}} = 1.$$

Where $(f_t)_{max} = \mu N$ and $(m_z)_{max}$ is the maximum moment alongside z – axis for the point of contact.
A further detailed explanation on the same is provided in the papers cited in the references section.

Hence, by ensuring all the elements in the matrix satisfying the above-mentioned condition, the 'H matrix' is constructed as:

$$\begin{matrix} fx,l & fx,r & 0 \\ fy,l & fy,r & 0 \\ mz,l & mz,r & 1 \end{matrix}$$

Where:

- f x,l is the force along x – axis at left end
- f y,l is the force along y – axis at left end
- mz,l is the moment along z – axis at left end
- f x,r is the force along x – axis at right end
- f y,r is the force along y – axis at right end
- mz,r is the moment along z – axis at right end

LTV MPC Formulation:
The idea behind the MPC formulation is to optimize the future behavior across a finite prediction horizon of p steps. At every discrete-time instant k, for a given state estimate e[k], the optimal control input is computed solving the following constrained quadratic programming [5]. The mathematical model for the same (described in further details in the papers as cited) is given as a Quadratic Programming:

$$\begin{aligned} \min_{z_k} \quad & J(z_k, e[k]) \\ \text{s.t.} \quad & Mz_k < b, \\ & u_m[k+i] \le u[k+i] \le u_M[k+i], i = 0 \dots p \\ & \\ & \Delta u_m[k+i] \le \Delta u[k+i] \le \Delta u_M[k+i], \\ & i = 0 \dots p-1 \end{aligned}$$

where

$$z_k = \left[u[k]^T \, f_c^T[k] \, u[k+1]^T \, f_c^T[k+1] \, \dots \right. \\ \left. \dots \, u[k+p-1]^T \, f_c^T[k+p-1]\right]^T$$

Fig12. Quadratic Programming for LTV-MPC

Quadratic Programming is the decision variable containing the input vector u[k+i] (that collects the linear and angular acceleration of the robots) and fc [k + i] (that collects the forces imposed on
the pushed object) for i = 0, . . ., p – 1. Such a definition of the decision variable allows us to consider the physical limitations of the robot inside the QP problem, even though the contact forces are not considered in the robot model. Besides, the cost function is defined as the following quadratic function:

$$J(z_k, e[k]) = \sum_{i=0}^{p-1} \{[e[k+i]^T Q e[k+i]] + [u[k+i]^T R_u u[k+i]] \\ + [\Delta u[k+i]^T R_{\Delta u} \Delta u[k+i]]\} \\ + e[k+p]^T P e[k+p].$$

Fig13. Optimal Cost Function for LTV-MPC

A further detailed explanation on the same is provided in the references as cited. Hence, the pseudocode which the algorithm is roughly based on for using MPC for robot traversal is:

send previous (v_r, ω_r)
get $e[k], v[k]$
$\tilde{e}[k] \leftarrow$ predict($e[k]$,$v[k]$,z_{k-1})
$A_k, B_{u_k}, B_{v_k}, M \leftarrow$ linearize($\tilde{e}[k], v[k]$)
$z_k \leftarrow$ QP($A_k, B_{u_k}, B_{v_k}, M, \tilde{e}[k], v[k]$)
$(v_r, \omega_r) \leftarrow (v_r, \omega_r) + T_s u[k]$
$z_{k-1} \leftarrow z_k$

Fig14. Pseudocode for LTV-MPC

Pushing Constraints for Object Slippage Avoidance:
Since the robot is subject to nonholonomic constraints, it cannot change its orientation instantaneously. The direction of the force applied to the pushed object is thus constrained as well. Besides, as previously discussed, the pushing force must be restricted within the friction cone to avoid object slippage during manipulation. Hence, the paper introduces a constraint for robot motion, such that the contact between the robot and the object does not break [5]. This guarantees that the movement of the robot produces valid pushing forces, that lie within the friction cone. The concept was applied as follows:

$$\begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \end{bmatrix} + \omega_r \times R(\theta_r) p_{or} = \begin{bmatrix} \dot{x}_o \\ \dot{y}_o \\ \dot{\theta}_o \end{bmatrix} = HGf_c$$

Fig15. Pushing Constraint for LTV-MPC

Where p_{or} is the position of the object in the robot frame Σr, R(θr) is the rotation matrix between Σr and Σw, ω_r is a 1 x 3 matrix representing angular velocities of the robot along each of the axes, x'_r, y'_r, θ'_r are the linear and angular velocities of the robot, x'_o, y'_o, θ'_o are the linear and angular velocities of the object,
H, G and f_c are the same matrices as discussed above.

V. IMPLEMENTATION

A. Virtual Dipole Fields

For the initial stages of implementation attempts, we had only considered linear motion for the robot and aimed at

observing the motion of the robot w.r.t the object around which the dipole field was created in a linear, one – dimensional setup along either of the X and Y axes. The result for the same was shown to be favorable as the dipole thus generated around the object ensured a smooth movement for the robot such that it always aligns itself parallel to the object's orientation, thus showing the effectiveness of the virtual dipole field algorithm thus replicated.

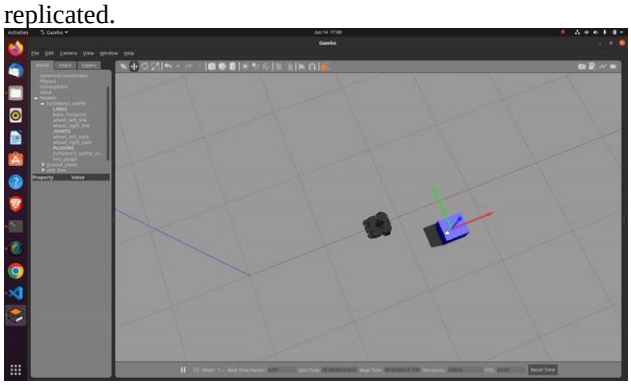

Fig16. Implementation for Virtual Dipole Field

We further went ahead to the consideration of an angular velocity for the robot to implement a more complicated scenario of the motion of the robot around the dipole field thus generated around the target object. As a result, we could achieve a better orientation for the robot to achieve w.r.t the orientation of the target object to be manipulated.

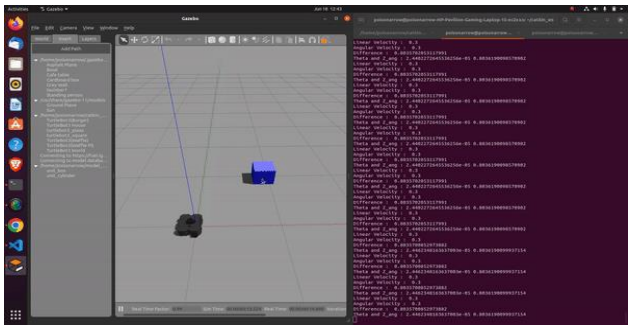

Fig17. Box pushing by Virtual Dipole Field

The final output of the implementation showed partially successful results for the desired object manipulation, though in a few specific cases of the implementation, the nature of robot was to often loose contact of the target object in turns and then reorient itself with the desired direction of motion again, by the influence of the dipole field thus generated.

Drawbacks:

- The algorithm makes a constant assumption of the fact that the position and the orientation of the target object is constantly known in order to create a frame of reference for the robot's orientation in the same.
- The algorithm makes no such consideration of the frictional forces acting between the object and the robot, which might later be a problem considering the varying material properties of objects.
- The algorithm makes no such consideration of the shape and size of the object, which might later be a problem considering the varying material properties of objects.
- A few specific cases for the manipulation often makes the robot loose contact of the object, thus making the robot go into recovery of orientation often under the influence of the virtual dipole field thus implemented.

B. Linear Time Varying MPC

To achieve the maximum similar results as possible for our implementation, the default URDF files of the TurtleBot3 Waffle model were edited to further add a rectangular flat plate on its surface. Furthermore, it was essential to get constant feedback from the pushing interaction between the object and the robot to map the amount of force being exhibited on the object to be manipulated by the robot. Hence, force – torque sensors were also added to both of the links on the sides of the rectangular plate to get an output of the force the object was being pushed with.

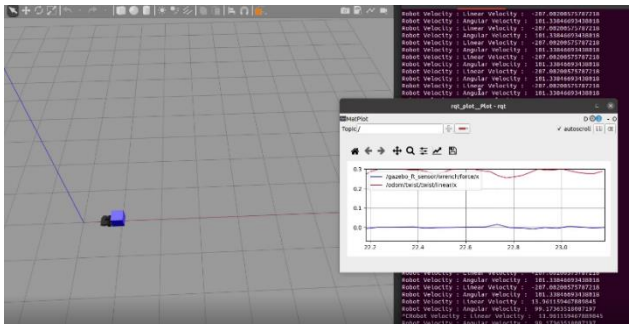

Fig19. Force-Torque outputs by the robot

For the implementation of MPC controller, Python3 libraries named as '*casadi*' and '*do_mpc*' were used in order to handle the backend calculations of the LTV MPC formulated controller. The result for the same ensured the robot to go to a specified location in the similar above-mentioned algorithm based on optimization of the state of the robot on the basis of predicted future states for the robot, in order to make it reach the final position as specified.

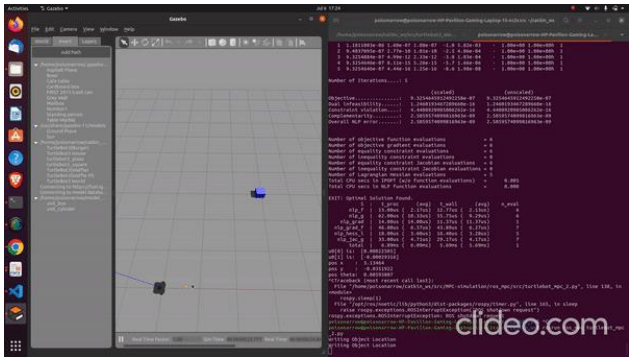

Fig20. MPC Controller Formulation for Robot Traversal

The slippage avoidance constraint as discussed above, was coded in Python3 by using the appropriate mathematical tools and python libraries, the resultant velocities for the robot derived from the same was sent as feedback to the LTV - MPC controller in attempts to perform a non – prehensile manipulation on a straight line for the object. The resultant output thus obtained proved to be quite robust as the relative slip of the object and robot, in motion was almost negligible, until a certain high threshold velocity of the TurtleBot3 which is quite impractical to attain in the real-world implementation of the same.

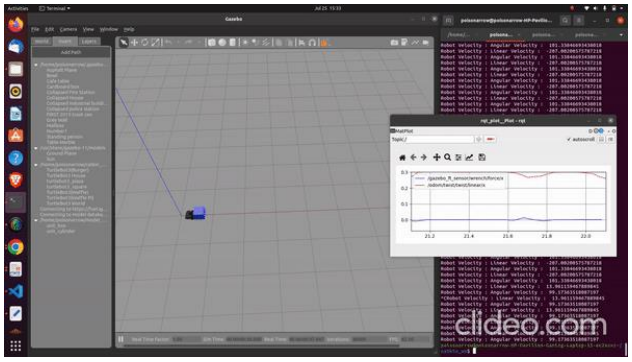

Fig21. Slip Avoiding constraints for LTV-MPC

VI. EVALUATION RESULTS

A. A* Search Result

The robot was successfully able to traverse the optimal path available in the environment simulated in gazebo, from the initial state to the goal state. The approach took in consideration of all the available static obstacles while calculating the optimal path to goal.

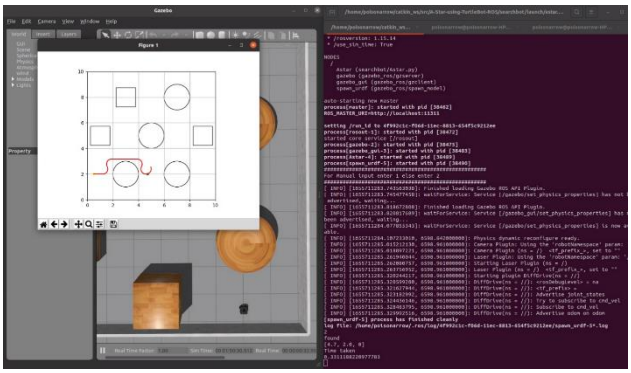

Fig22. A Search Result for TurtleBot3*

B. RRT Path Plan Result

The robot was successfully able to traverse the optimal path available in the environment simulated in gazebo, from the initial state to the goal state. The approach took in consideration of all the available static obstacles while calculating the optimal path.

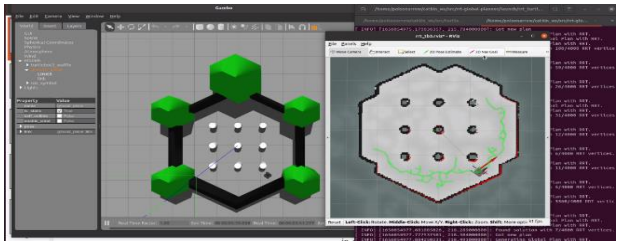

Fig22. RRT Implementation for TurtleBot3

C. Algorithm Evaluation

In order to judge the robustness of both the algorithms it is therefore necessary to expose both these algorithms to physically varied environments in order to see their robustness with varied types of object interaction. To keep our test results closest to being applicable in real world, the robot model to work with was chosen as same in all cases performed, i.e the standard rectangular waffle model of the Turtlebot3 robot with the additional change of an attached plate equipped with Force-Torque sensors.

All the cases thus described in the images and the evaluation table presented below, were also tested with varied lengths, dimensions and radius of the box and cylinder respectively and similar nature of behavior was thus observed in all the varied cases (while having minor deviations in the case of Virtual Dipole Field). Additionally, the coefficients of friction were majorly kept having three values each representing scenarios of **less, moderate and maximum** possible friction on the interacting surfaces in order to efficiently get an idea of how the algorithms shall work when varied by the same.

For the sake of analysis simplicity we had designed the controllers in such a way to keep following a predefined straight path overtime and observe the nature of pushing for both these algorithms.

Object Type	μ	Virtual Dipole	LTV-MPC
Box	0.1	Object Slipped	Object Push Worked
Box	0.5	Object Slipped	Object Push Worked
Box	1	Object Push Worked	Object Push Worked
Cylinder	0.1	Object Push Worked	Object Push Worked
Cylinder	0.5	Object Push Worked	Object Push Worked
Cylinder	1	Object Push Worked	Object Push Worked

Fig23. Evaluation Table

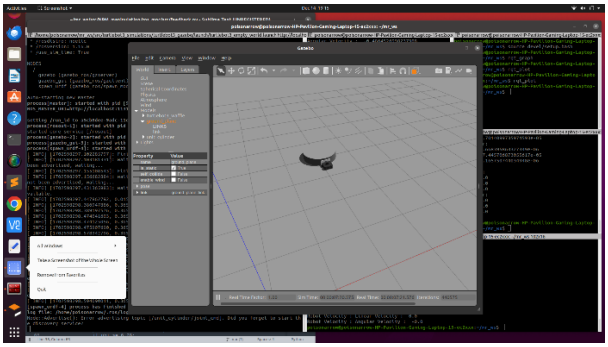

Fig24. Object Pushing for a Cylinder

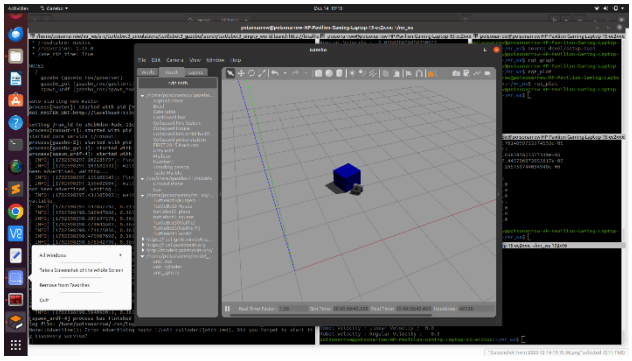

Fig25. Object Slipping Contact for a Box

Hence. It can be concluded that LTV-MPC thus produces more effective results for pusing mechanisms to be implemented primarily due to the important consideration of a robust system dynamics.

VII. CONCLUSION

The current status of the codebase, aimed at replicating some of the most efficient and notable research done in the field of '*non – prehensile manipulation by wheeled robots*', is kept with two different approaches. The first approach is a '*virtual potential field'* based approach where an algorithm to push an object over a flat surface is implemented by creating a virtual dipole field around the object to be manipulated in order to decide the motion of the robot around it. The input for the same is heavily dependent on the assumption of the pose and location of the target object to be known at all times. In our replication results for the same, we have indeed achieved partial success for most of the object pushing scenarios. However, it is also observed that this approach often makes the robot loose contact of the object and the algorithm makes no consideration of the frictional forces acting between the object and the robot along with the shape of the object being manipulated, which proves to be a major drawback for the approach.

In order to overcome the above referred problems a new approach was formulated, inspired from the *LTV – MPC* formulated controllers for robot motion, along with an added constraint to ensure minimal slippage between the robot and the object being manipulated while taking the due deliberation of the frictional forces acting between the robot and the object. It was also proved that the maximum efficiency in achieving the same is possible through an '*Ellipsoidal Limit Surface*' being implemented for the interactive forces between the surfaces of the robot and object. Hence, the object slippage avoidance constraint was implemented by taking in thoughtful consideration of the same. The final implementation showed positive results for a straight-line motion of non – prehensile manipulation till a specific high threshold velocity of the robot which is practically unattainable in a real-world scenario.

The future scope of the above referred work done holds on to the potential of optimizing the current existing result by certain *trajectory optimization* techniques, in order to achieve better results shown by the above referred papers. The necessary literature survey is being conducted by the team to achieve the same.

REFERENCES

[1] Kevin M. Lynch, et. al. "Stable Pushing: Mechanics, Controllability, and Planning" 1999 The International Journal of Robotics Research, 15(6):533–556, 1996.

[2] Kevin M. Lynch, "Modern Robotics"

[3] C. Urmson and R. Simmons, “Approaches for heuristically biasing rrt growth,” in Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), vol. 2. IEEE, 2003, pp. 1178–1183

[4] T. Igarashi, Y. Kamiyama and M. Inami, "A dipole field for object delivery by pushing on a flat surface," *2010 IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, 2010, pp. 5114-5119, doi: 10.1109/ROBOT.2010.5509483.

[5] F. Bertoncelli, F. Ruggiero and L. Sabattini, "Linear Time-Varying MPC for Nonprehensile Object Manipulation with a Nonholonomic Mobile Robot," 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 2020, pp. 11032-11038, doi: 10.1109/ICRA40945.2020.9197173.

[6] Fakhari, A, Keshmiri, M, & Keshmiri, M. "Dynamic Modeling and Slippage Analysis in Object Manipulation by Soft Fingers." *Proceedings of the ASME 2014 International Mechanical Engineering Congress and Exposition*. *Volume 4A: Dynamics, Vibration, and Control*. Montreal, Quebec, Canada. November 14–20, 2014.

[7] Suresh Goyal, Andy Ruina, Jim Papadopoulos,
Planar sliding with dry friction Part 1. Limit surface and moment function, Wear, Volume 143, Issue 2,
1991, Pages 307-330, ISSN 0043-1648, https://doi.org/10.1016/0043-1648(91)90104-3.