Real-Time Evolutionary Model Predictive Control Using a Graphics Processing Unit

Phillip Hyatt  Marc D. Killpack

Abstract—With humanoid robots becoming more complex and operating in un-modeled or human environments, there is a growing need for control methods that are scalable and robust, while still maintaining compliance for safety reasons. Model Predictive Control (MPC) is an optimal control method which has proven robust to modeling error and disturbances. However, it can be difficult to implement for high degree of freedom (DoF) systems due to the optimization problem that must be solved. While evolutionary algorithms have proven effective for complex large-scale optimization problems, they have not been formulated to find solutions quickly enough for use with MPC. This work details the implementation of a parallelized evolutionary MPC (EMPC) algorithm which is able to run in real-time through the use of a Graphics Processing Unit (GPU). This parallelization is accomplished by simulating candidate control input trajectories in parallel on the GPU. We show that this framework is more flexible in terms of cost function definition than traditional MPC and that it shows promise for finding solutions for high DoF systems.

I. INTRODUCTION

As humanoid robots become more complex and widely used, there is a need to control higher degree of freedom systems in more robust ways to accommodate a variety of environments and disturbances. Model predictive control (MPC) has been shown to be robust to modeling errors and disturbances while controlling compliant and underdamped robots [1]. However, applications for low-level (or high rate) humanoid control using MPC are currently limited by the computational speeds associated with long prediction horizons and large numbers of degrees of freedom (DoF). This work seeks to address the issue of MPC scalability as well as enabling further research in robust MPC through the parallelization of the optimization required for MPC.

Several works have proposed the use of parallelization to increase the speed of MPC. In [2] the authors develop simplifying assumptions about the model to decrease the search space of the optimization. These assumptions decouple the inputs and therefore optimizations for each input can be solved in parallel as in [3]. Others have proposed using parallelism to increase the speed of gradient based optimizers used in MPC as in [4] and [5] or splitting up the optimization into several parts solved in parallel as in [6]. In [7], using a Graphics Processing Unit (GPU), disturbances are simulated on a set of a few inputs to find the inputs that are the most robust to these disturbances. In our case, we do not introduce disturbances, but simulate sets of many control inputs to find the control trajectories that decrease a user defined cost function. This is most similar to the work done in [8], [9], where they derive a method that guarantees optimal control input sampling based on their formulation. However, the real hardware platforms controlled in these papers have few states and control inputs and they seem to be focused on mid to high-level controllers running at about 40 Hz. Whereas we are developing low-level controllers running at greater than 350 Hz. In [8] the approach also required a empirically-based, manually tuned model and [9] required 30 minutes of expert driving to learn a sufficiently accurate model from which to sample. In addition, they assume that noise is only present on the control inputs which makes sense for off-road racing but not for robot arm control.

Although the formulations in [8], [9] are different than the one presented in this paper, it is too early in the development of GPU-based MPC to determine which is better (or better in certain situations). A direct comparison, although important, is out of the scope of this paper and is left for future work. However, we do compare against another gradient-based MPC method instead. Work has been done to study the effects of simplifying assumptions in the dynamics of humanoid robot arms [10]. These simplifying assumptions can be used to greatly reduce the complexity and dimensionality of the optimization problem in MPC making it tractable with the use of gradient based solvers for high-rate control. We use the results presented in [10] for comparison in Section III.

A sampling based approach to the optimization in MPC for low-level control of humanoid robots has previously been infeasible because of the computation time needed to sample the entire input space. This input space can be very high dimensional since it is a function of the degrees of freedom in the input and the length of the time horizon. Our implementation of EMPC has only been made possible in recent years by the emergence of massively parallel computing architectures such as GPUs. Additionally, this work outlines several simplifying assumptions about the input space which effectively shrink it to a reasonable search area. We believe that there is still much research to be done in making this search more efficient as discussed in Section V, but this work provides evidence that EMPC can be run at real-time rates using existing hardware on humanoid robot manipulators with fast and underdamped dynamics.

While this work will not explore all of the potential benefits of EMPC, several are readily seen. Because no gradients are calculated, the cost function used in the optimization need not be convex or even continuous to allow for solution. This allows great flexibility in constructing costs, including costs with conditional statements such as “if” or “else.” Moreover, as costs of several inputs are calculated at once, gathering statistical measures of robustness to input disturbances would...
be straightforward and could be included in a larger overall cost aimed at finding robust control input solutions. Lastly, because power consumption has replaced transistor size as the limiting factor in computation speed [11, microprocessor manufacturers are increasingly turning to parallelization for performance increases [12]. As the amount of parallelization in computer hardware increases, the size and complexity of EMPC control problems will be able increase along with it.

The main contributions of this paper are as follows:

- A method for implementing real-time EMPC on a GPU is presented.
- Real-time control of a 7 DoF compliant robot using EMPC is simulated.
- Real-time control of a 7 DoF compliant robot using EMPC is performed on real hardware at speeds of greater than 350 Hz. The performance of this controller is compared with other model predictive controllers on the same hardware.

In terms of paper organization, we explain the models and methods used to implement EMPC using a GPU in Section II. In Section III we outline the experiments carried out in simulation and on real hardware, and in Section IV we present and discuss the results from these experiments. Section V outlines future applications and research directions possibly derived from this work.

II. METHOD

Because EMPC relies heavily on the modeling and simulation of a dynamic system, this section will first explain how the modeling and simulation was accomplished for this paper. We then present the EMPC algorithm and explain its implementation on an NVIDIA GPU using the CUDA programming interface. The nomenclature used in this paper is defined in Figure 2.

A. Dynamic Model and Simulation

The dynamic model used in this work is defined by Eqn 1 which is for a serial manipulator with rigid links and no applied external forces (see [13]):

$$\tau = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + F(\dot{q}) + G(q)$$  \hspace{1cm} (1)

Neglecting friction and assuming that a low level controller compensates for gravity, (which is the case for the Baxter robot from Rethink Robotics used in this work) this becomes:

$$\tau = M(q)\ddot{q} + C(q, \dot{q})\dot{q}$$  \hspace{1cm} (2)

We choose to apply torques given by a low stiffness impedance controller to make this robot safer in human environments [14] by implementing the torque control $\tau$ as follows:

$$K_p(q_{cmd} - q) - K_d\dot{q} = M(q)\ddot{q} + C(q, \dot{q})\dot{q}$$  \hspace{1cm} (3)

Solving for $\ddot{q}$ to put the equations in state variable form gives:

$$\ddot{q} = M(q)^{-1}[K_p(q_{cmd} - q) - K_d\dot{q} - C(q, \dot{q})\dot{q}]$$  \hspace{1cm} (4)

We can then group terms as follows:

$$A = \begin{bmatrix} M(q)^{-1}(-K_d - C(q, \dot{q})) & M(q)^{-1}(-K_p) \\ 1 & 0 \end{bmatrix}$$  \hspace{1cm} (5)

and

$$B = \begin{bmatrix} M(q)^{-1}K_p \\ 0 \end{bmatrix}$$  \hspace{1cm} (6)

we can represent this system in state variable form as

$$\begin{bmatrix} \ddot{q} \\ \dot{q} \end{bmatrix} = A \begin{bmatrix} \dot{q} \\ q \end{bmatrix} + Bq_{cmd}$$  \hspace{1cm} (7)

where Eqns 5 and 6 are nonlinear functions of the state. Eqns 5 and 6 were evaluated using an open source symbolic robotics library [15], [16]. Given initial states and an input trajectory ($q_{cmd}$), this system of nonlinear differential equations can be integrated to simulate the robot’s behavior. As a first approximation, the simulations performed in this work are all accomplished using first order integration as defined in Eqn 8 where $\ddot{q}_i$ can be evaluated at each time step using Eqn 7:

$$\begin{bmatrix} q_{i+1} \\ \dot{q}_{i+1} \end{bmatrix} = \Delta t \begin{bmatrix} \ddot{q}_i \\ \dot{q}_i \end{bmatrix}$$  \hspace{1cm} (8)

There are two different types of simulation carried out in this work. The first type is performed on the GPU, assumes $A$ and $B$ are constant over the horizon, and is simulated with $\Delta t = .002$s. $A$ and $B$ are assumed constant over the horizon because these matrices are actually state dependent and recalculating them for each individual simulation would be computationally costly. This is essentially a method of linearization which gives good results near the initial states. In addition, although this is something that we may modify in future work, this method has proven effective in past work.
where MPC was used to control a robot manipulator with compliance at the joints [17], [18].

The second type of simulation is carried out on the CPU for the nonlinear real-time simulation in section III. This simulation uses $\Delta t = .00001s$ and updates the $A$ and $B$ matrices at each step in order to provide a more accurate representation of the dynamics.

### B. EMPC implementation on a GPU

GPUs are designed to launch thousands of threads which all perform the same function on different data. In the CUDA programming interface these threads are assigned unique indices and are organized in structures called blocks as seen in Figure 3. This hierarchy is important to understand because only threads within a block can be guaranteed to be run synchronously. These considerations are important when simulating and evaluating costs in parallel on a GPU.

As will be explained in more detail, EMPC utilizes the parallel computing capability of GPUs by evaluating the fitness of hundreds or thousands of input trajectories simultaneously. For our evolutionary algorithm, parent input trajectories are selected as either the best input trajectories from a previous generation or from a random uniform distribution (in the case of a cold start). These parent trajectories are then “mated” to produce children which are then simulated on the GPU. This process of mating and simulating is repeated either until some convergence criteria are satisfied or until a set number of steps ($K$) have been reached. The number of evolution steps $K$, number of parents $P$, number of parents per child $P_{perchild}$, and the number of mutations $N_{mutations}$ are all tunable parameters of the optimization. The EMPC algorithm is represented in Algorithm 1 and is explained in detail in the following sections.

1) **Cold Start Stochastic Trajectory Generation:** Before an input trajectory can be simulated on a system, the input trajectory must be created or defined on the GPU. In order to decrease the size of the search space without sacrificing horizon length, a trajectory for each input is parameterized with $M$ evenly spaced control points $c$ joined together with cubic splines as defined in [19] and depicted in Figure 4.

This idea is similar to that of the Move Blocking (MB) strategy often used to simplify MPC problems, which has even been parallelized in [20]. A key difference is that MB constrains inputs to be constants over portions of the time horizon, while using splines still allows for dynamic inputs over the whole horizon. In this way the input trajectory is continuous and there is an intuitive way of bounding the search area. For example, the optimal inputs can be assumed to never go outside some bound surrounding the steady state inputs. This form of parameterization also lends itself well to warm starting a solution with past optima.

The search space dimension can be further decreased by considering limitations of actuation in real dynamic systems. For example, because the motors in our robot cannot output torque discontinuously, we define the first point in our input trajectory to be the last commanded input to the real system. Other restrictions can be made to further decrease the search space, such as constraining the final point to be the steady
Algorithm 1 Evolutionary Model Predictive Control

1: if Cold Start then
2:   Given \(q_{cmd_{i-1}}\) and \(q_{goal}\)
3:   Define \(q_{cmd}\) using eqns 9-12
4: else if Warm Start then
5:   Define \(q_{cmd}\) as \(Child\) trajectories from last solve
6: end if
7: for \(i = 1\) to \(T\) do
8:   In parallel for each DoF:
9:     \[ \ddot{q}_{i+1} = M(q_i)^{-1}[K_p(q_{cmd_i} - q_i) - K_d \dot{q}_i - C(q_i, \dot{q}_i) \dot{q}_i] \]
10: \[ \dot{q}_{i+1} = \dot{q}_i \Delta t \]
11: \[ q_{i+1} = q_i + \dot{q}_i \Delta t \]
12: \[ J = J + \text{integral cost} \]
13: end for
14: Select \(P\) lowest \(J\)s and associated \(q_{cmd}\) trajectories
15: \(Number of matings = P - P_{perch} + 1\)
16: for \(i = 0\) to \(Number of matings - 1\) do
17:   \(Child[i*Number of mutations + j] = \text{best DoF trajectories from parents}\)
18:   Mutate \(Child[i*Number of mutations + j]\)
19: end for
20: end for
21: Repeat Steps 1-22 \(K\) times
22: Apply first input from best child

\[ c_{j,M} = q_{goal}[j] \] \hspace{1cm} (10)
\[ \sigma_{init,j} = .5|q_{goal}[j] - q[j]| \] \hspace{1cm} (11)
\[ c_{j,k} \sim \mathcal{N}(q_{goal}[j], \sigma_{init,j}^2) \quad \forall k \in 2...M - 1 \] \hspace{1cm} (12)

In the case of a warm start, instead of choosing the \(P\) best control inputs from a random distribution to be the parents, the \(P\) parent trajectories from the past optimization are used as parents for the next generation. This is described in Algorithm 1.

In both warm and cold starts, we define one of the input trajectories as a step input to the final goal state (all control points are chosen to be \(q_{goal}\)). This serves to force the solutions of each generation toward the goal and also ensures that when the system has reached the goal states, it is less likely to deviate from them.

2) Simulation and Cost Calculation: The simulation of an input trajectory over a horizon is done sequentially using Equation 8 where the simulation of every coupled degree of freedom during step \(i\) must be done before simulating the next time step \((i+1)\), therefore this part of the algorithm becomes the computational bottleneck. To maximize the parallelization at this step, several full-arm control trajectory simulations are run in parallel in the same block. Each individual thread in a block does calculations for the angular acceleration \((\ddot{q})\) for 1 DoF as seen in Figure 3. There is a limit to the number of threads per block (1024), so this means that for our system with 7 DoF the maximum number of parallel full-arm simulations per block is 146. Several blocks may be launched at once however, allowing for thousands of simultaneous simulations. The effect of having more processing cores on a GPU would be to allow for more blocks to be executed in parallel. This could mean faster performance if blocks are being queued one behind another or the ability to search a larger input space in the same amount of time.

Because threads in the same block can access the same shared memory and can be made to all finish step \(i\) before any thread starts step \(i+1\), each thread can update the states associated with its degree of freedom and have access to all other updated states throughout the simulation.

As mentioned before, the formulation of a cost in this framework is extremely flexible. The cost we choose for our implementation is defined by the following equations:

\[ J_{i,j} = \alpha \frac{q_i[j]}{1 + |q_i[j] - q_{goal}[j]|} + \beta |q_i[j] - q_{goal}[j]| \] \hspace{1cm} (13)
\[ J = \sum_{j=1}^{N} \sum_{i=1}^{T} J_{i,j} \] \hspace{1cm} (14)

where \(\alpha\) and \(\beta\) are scalar weightings on velocity near the goal and position error respectively and \(i\) and \(j\) are iterators.

---

Fig. 4: Spline trajectory for the sample control points.
for time step and DoF. The first term in the cost drives $\dot{q}$ to be low when near the goal and the second term drives $q$ toward the goal. The added $+1$ in the denominator of the first term ensures numerical stability (avoiding division by zero) when $q$ approaches $q_{\text{goal}}$. Because each thread calculates the cost associated with its DoF we really have access to $N$ costs, but sum them all to get a $J$ value (see Eqn 14) representative of the whole system. These $N$ costs associated with each DoF, however, will be used as part of the crossover process explained in section II-B.4.

3) Parent Trajectory Selection: For this work, a very simple method of choosing the parent control trajectories is used. While reading the costs $J$ associated with each simulation from the GPU, the CPU maintains the $P$ lowest costs and their associated trajectory control points in an ordered array. The total cost of the trajectory is saved, as well as the cost associated with each individual DoF. In future work this sort may be done on the GPU using a parallelized sorting method such as merge or radix sort as in [21].

![Fig. 5: Example of one full generation with $P = 5$, $P_{\text{perchild}} = 3$, and $N_{\text{Mutations}} = 2$](image)

4) Crossover and Mutation: In our implementation of the evolutionary algorithm, we are not limited to 2 parents per child, but rather can choose any number of parents per child which is less than or equal to the total number of parents. The numbers of parents per child ($P_{\text{perchild}}$) and total number of parents ($P$) determine the number of matings. This is described graphically in Figure 5 and mathematically as:

$$\text{Number of matings} = P - P_{\text{perchild}} + 1; \quad (15)$$

For each mating, the children trajectories are produced by crossover of the $P_{\text{perchild}}$ parents. For crossover, we simply look at the $P_{\text{perchild}}$ parents and take the trajectory with lowest associated cost for each DoF. In this way, each child is formed from the best parts of all of its parents. While each of these DoFs are in reality coupled to all of the others through the system’s dynamics, the approach of treating them independently seems to lead quickly to convergence. Also, because we are choosing the best parts of many parents in each mating, the children are more likely to converge to good solutions quickly while the mutation step prevents stagnation in local minima.

As seen from the right side of Figure 5, from each mating the number of children produced is simply equal to the number of mutations $N_{\text{Mutations}}$, which is a tuning parameter for the optimization. It is clear that to achieve more mutations in this framework there will need to be more simulations, which means more computation for each generation. This is an important trade-off for an application to real-time MPC because of the need for fast solutions.

The mutation step is performed after the crossover step. In our case we simply perturb the control points for the input spline by small random numbers.

III. Experiments

A. Hardware and Setup

The robot used in this paper is a Baxter Research robot manufactured by Rethink Robotics. A high rate torque controller is used to enforce a low stiffness impedance controller running at 500 Hz. This system has been intentionally tuned to allow for greater compliance for safety in case of incidental contact. Communication with the robot for sending inputs and recording joint angles was done through the Robot Operating System (ROS).

The computer used for this work is an HP desktop computer with an i7-4770 3.40 GHz processor and an NVIDIA GeForce GTX 750 Ti GPU with 640 cores.

![Fig. 6: ROS node and topic diagram for implementation of EMPC](image)

We chose to implement and run our algorithm using ROS (Robotics Operating System) as is depicted in Figure 6. EMPC was implemented as a ROS service which receives the goal states, current states, and a boolean indicating warm or cold start and returns the control points for the best solution it found. After sending and receiving information from the EMPC service, the Baxter controller node simply translates the received control points into an input trajectory and then publishes the first input to the Baxter simulation or to an impedance controller connected to hardware.

All experiments were done with a model prediction horizon of 150 time steps and the controller was allowed to run as fast as it could solve.

B. Real-time EMPC in Simulation

The Baxter simulation node used the inputs from the $q_{\text{des}}$ topic to simulate the states of the robot at a high rate as explained in section II-A. In order to make this simulation
even more realistic, noise was added to the states before they were published to the Baxter controller node. The noise was randomly distributed with a mean of zero and standard deviations of .1 and .01 for $\dot{q}$ and $q$ respectively. See Figures 7 and 8 for examples of the amplitude of the noise.

Fig. 7: Joint angle response of 7 degrees of freedom using a step input in real-time simulation.

Experiments were conducted to assess the ability of the EMPC controller to reduce settling time, overshoot, and rise time for this underdamped system. Starting from a zero configuration, the EMPC controller was given a step input of 1 radian for every joint simultaneously. For comparison, this same experiment was carried out without the EMPC controller, so the step input was sent directly to the low level impedance controller. Joint angle data as well as impedance controller inputs were recorded.

C. Real-time EMPC on Hardware

Building on the simulation experiment, a similar test was done on real hardware using the same ROS Service call to do EMPC. The only difference for this experiment was that a low level impedance controller was run at 500 Hz which subscribed to inputs from the Baxter controller node and converted them to torques applied to the robot.

This experiment was conducted to test the robustness of EMPC with regard to modeling error. The model used for all of the simulations in this work is based on parameters given by the manufacturer with no efforts at improvement. In addition, friction was completely excluded from our model. While these flaws in the model are known and methods exist to minimize them, we prefer a controller which is able to perform well despite small to moderate modeling error.

The ultimate test of our controller was to use it to make large movements and to observe whether or not overshoot and settling time were reduced without large increases in rise time. To test this we commanded sequential step changes of .5 and then -1 radians to all joints simultaneously. The results for these tests using a step input and using EMPC can be seen in Figures 10 and 11 (and in the accompanying video for this paper).

D. Comparison with Gradient Based MPC

Because a step input to the torque controller is the simplest and most naive approach, it was also necessary to compare our EMPC algorithm with comparable gradient-based model predictive controllers. For this experiment, results were compared with those of [10] in a movement of over 1 meter in end effector position. Except for the goal joint positions, this experiment was conducted exactly as in section III-C.

IV. RESULTS AND DISCUSSION

A. Real-time EMPC in Simulation

Figure 7 shows the joint angle response in simulation to a step input of 1 radian to every joint. This response is typical of an underdamped system with coupling between the degrees of freedom. As expected, there is more overshoot and longer settling time for the more proximal joints because they have greater inertia than the more distal joints.

Figure 8 shows the joint angle response when using the EMPC algorithm. Average rise times, settling times, and overshoots were computed across all joints and are represented graphically in Figure 9. As can be seen, EMPC succeeded in significantly reducing overshoot and settling time, while increasing rise time only slightly. While the level of noise on the position and velocity were well above what
Fig. 9: Box and Whisker Plots comparing Rise Time, Settling Time, and % Overshoot for step input and EMPC input in simulation and on real hardware over all 7 DoF of the robot. The bottoms and tops of boxes represent the first and third quartiles respectively. Whiskers represent the rest of the data while hollow and filled circles represent outliers and median values respectively.

we would expect from a real platform, the EMPC algorithm demonstrated robustness to this noise.

It should also be noted from the Figures that joint 7 does not behave as an underdamped joint in any of the simulations. This has to do with the very small inertia associated with this joint. It is also interesting that EMPC does not increase the rise time of this joint.

B. Real-time EMPC on Hardware

This hardware experiment showed that the EMPC algorithm was able to run in real-time for 7 DoF and was able to significantly reduce overshoot and settling time when compared to a simple step input with an imperfect model of the dynamics. The controller rate was measured to be 372 Hz with a standard deviation of .00028s and a max solve time of .006s. The comparison of simulated and actual results for the step input case is a good indicator of the model accuracy. One limitation of our current implementation is the obvious steady state error on real hardware. Steady state error in the real hardware is most likely due to imperfect gravity compensation torques (calculated using manufacturer given parameters) or unmodeled joint friction. In addition, the low-level impedance controller is a simple Proportional Derivative (PD) controller and includes no integral control (see Section II-A). It should also be noted that joint 4 has a limited range on the hardware (0-.5 rad) that was not enforced by the simulation which explains the discrepancy between the simulation and real hardware tests.

C. Comparison with Gradient Based MPC

The last hardware experiment compared the performance of EMPC with that of 2 other model predictive controllers. All of the controllers succeed in reducing overshoot and settling time in a large movement when compared to a simple step input as can be seen in Figure 12. From this figure it can be seen that although EMPC uses a sampling based optimization method, its performance is comparable to those of gradient based MPC approaches.

This is an encouraging result as it demonstrates that
although the evolutionary algorithm is not able to guarantee convergence to even a local optimum, it finds solutions that are comparable to those found by gradient based solvers.

V. Conclusion

Given our promising results, there is good reason to further explore applications and improvements on our method. Planned future work includes using more advanced integration methods for dynamic simulation on the GPU, using nonlinear models including Neural Networks to model system dynamics, and the use of a similar algorithm with longer horizons and solve times as a high level motion planner. The ability of this approach to scale to more degrees of freedom, as well as its application to underactuated systems would also be interesting topics of further research. Additionally, the ability to gather statistical data about control inputs could be useful for robust MPC algorithms.

In this paper we described, implemented, and tested in simulation and on a real robot a real-time EMPC controller. Results showed that EMPC is effective in reducing overshoot and settling time without greatly increasing rise time for an underdamped robot arm. This result is important in the control of compliant robots because it enables smooth, time-efficient movement of serial manipulators while still maintaining compliance for safety reasons.

ACKNOWLEDGEMENT

We gratefully acknowledge support from NASA ECF grant NNX14A051G which has made this research possible.

REFERENCES

[1] C. M. Best, M. T. Gillespie, P. Hyatt, L. Rupert, V. Sherrod, and M. D. Killpack, “A new soft robot control method: Using model predictive control for a pneumatically actuated humanoid,” IEEE Robotics & Automation Magazine, vol. 23, no. 3, pp. 75–84, 2016.

[2] K. V. Ling, J. Maciejowski, A. Richards, and B. F. Wu, “Multiplexed model predictive control,” Automatica, vol. 48, no. 2, pp. 396–401, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.automatica.2011.11.001

Fig. 12: Cartesian position error of the end effector using 3 separate MPC methods - EMPC (this work), and Cartesian and Disturbance-Torque MPC ([10]).