Adaptive Look-Ahead for Robotic Navigation in Unknown Environments

Greg Droge and Magnus Egerstedt

***Abstract*— Receding horizon control strategies have proven effective in many control and robotic applications. These methods simulate the state a certain time horizon into the future to choose the optimal trajectory. However, in many cases, such as in mobile robot navigation, the selection of an appropriate time horizon is important as too long of a time horizon can amplify the detrimental effects caused by environmental uncertainties in the prediction of the future state while too of a short horizon will lead to reduced performance. In this paper we strike a balance between these two conflicting objectives by first introducing a receding horizon method for navigation founded on schema-based behaviors. We then suggest a method of adapting the time horizon by minimizing a cost function which balances the performance of the underlying control problem (which prefers longer horizons) with the performance of our state prediction (which prefers shorter time horizons). We illustrate the operation with an example which shows the usefulness of our navigation scheme with an adaptive time horizon.**

I. Introduction

Robot navigation is the object of much research and the subject of many books (e.g. [1] and [2]). Many of these techniques simulate the actions of the robot into the future and choose the control based on some performance metric. Such methods range from search algorithms such as Dijkstra's algorithm and the many derivatives thereof (e.g. [1]) to optimal control solutions for generating appropriate trajectories (e.g. [3]). These methods are useful due to their ability to, given a cost function, choose the optimal control action for navigation while respecting constraints on the dynamics, control effort, and environmental factors.

However, one of the difficulties with navigating through a partially unknown environment is that the precomputed optimal trajectories can only be followed for a small amount of time. Over longer periods they can potentially produce problems caused, for example, by the occurrence of previously undetected obstacles. Receding horizon control strategies offer a potential remedy to this problem in that they utilize the usefulness of the optimal control strategies while at the same time adding a certain element of feedback into the system (e.g. [4]).

Instead of depending on the model to come up with the entire trajectory, receding horizon strategies only compute the trajectory over a given time horizon and then take a single step along that trajectory. This idea has proven highly useful in a number of different control applications (e.g. [5] and [6]). In robotics, this view is represented, for example, in [7] and [8]. *While receding horizon methods are able to capture the desirable benefits of optimal solutions, there is an inherent trade-off between using a large horizon to capture the optimal solution and a short horizon to decrease the detrimental effects of poor predictions.*

The typical approach in the receding horizon framework is to choose a fixed time horizon over which to predict the unknown variables and obtain the optimal control input. If we had perfect estimates we could then make the time horizon as large as possible subject to factors such as computation speed, convergence, stability, and satisfaction of terminal constraints (see for example [4] and [6]). However, when we do not have perfect predictions, a time horizon which is too large may be detrimental in that the effect of the poor estimate is amplified over a longer time period. Likewise, if we choose our look-ahead horizon to be too small then the solution may not count on the benefits of the underlying optimal control solution due to the fact that it does not look far into the future to see what is actually optimal.

The remainder of this paper will proceed as follows: In the next section we will introduce a receding horizon method for navigation based on schema-based behaviors. We explain the reason behind this approach and show the conditions for optimality. In Section III we will introduce a general framework for adapting the look-ahead horizon for receding horizon problems in an optimal fashion and give the optimality conditions. Then, in Section IV, we provide an example integrating our proposed receding horizon navigation control with an adaptive time horizon and end the paper with some concluding remarks.

Email: {gregdroge,magnus}@ece.gatech.edu.
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA.

II. Receding Horizon Schema-Based Navigation

Schema-based behaviors are founded on the idea that simple motor behaviors can be designed such that the aggregate behavior produces the desired result [2]. After the different behaviors are chosen, the overall movement of the robot is obtained by summing the outputs of the weighted behaviors [2]. However, assigning these weights is not a trivial task.

In navigation many behaviors are possible, such as move-to-goal, move-ahead, avoid-static-obstacles, stay-on-path, and avoid-past to name a few [2]. But how do you determine which of these is more important and assign weights accordingly? Unless there is some type of performance index, it will be up to the engineer to tune these weights until the desired behavior is observed, but this result would be highly dependent on the environmental variables at hand. Moreover, it is conceivable that the importance of each behavior could change over time, depending on the immediately surrounding environment.

There are methods for choosing and adapting these weights such as fuzzy-logic, genetic algorithms, and other learning procedures [2]. These strategies can be highly computationally intensive or require heavy tuning which would cause the solution to be heavily dependent on the specific environment. We propose that a receding horizon approach could be introduced to adapt these weights online by minimizing a performance index. This will allow the effect of the weights to be simulated into the future to choose the weights that would result in the lowest cost.

We choose to do this instead of using the cost function to determine the optimal trajectory directly. Computing the optimal trajectory can be a very computationally intensive problem because it often requires methods which can have poor convergence characteristics or require intense computation (e.g. [3]). To perform this optimization at every time step can be difficult (e.g. [5]). On the other hand, parameter optimization is much less computationally intensive and often, as in this case, only requires integrating the state forward in time and a costate backward in time for each calculation of the gradient.

Behavior-based schemas provide a good framework in which this parameter optimization can take place. It is assumed that one has designed or could design these behaviors to make the robot act in the desired fashion. This parameter optimization basically chooses which behavior combination is needed at the given time to perform the task.

A. Navigation Schemas

Navigation using schema-based behaviors is equivalent to using potential functions for navigation, except for the fact that in the potential functions method a potential function is defined and the movement of the robot is in the negative gradient (e.g. [10]) whereas the schema-based navigation directly defines the negative gradient. There are many more advanced potential functions than the ones presented here (e.g. [1], and [10]), but we have chosen our functions to demonstrate the utility in adapting both the weights and the time horizon.

We are going to assume that we have a robot with N distance sensors distributed evenly around the robot. These sensor measurements are scalar distance values from which we assume we can calculate the planar position of an obstacle point. We denote the point of the i^{th} sensor measurement as $o_i \in \mathbb{R}^2$. An example of this type of sensor measurement can be seen in Figure 1. It shows a robot with $N = 50$ sensors, each sensor with a limited range of sensing.

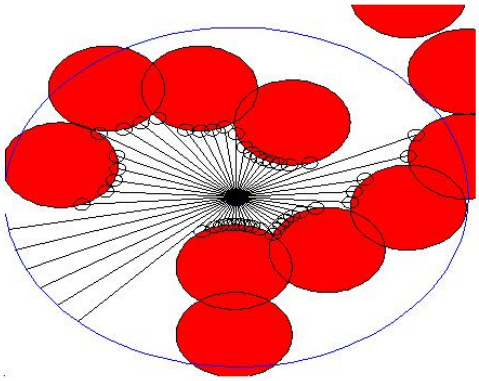

Fig. 1: Above shows an example of the distance sensing we will use. The large blue circle shows the sensing limit of the distance sensors while the small circles represent locations where the obstacles were sensed.

We let the state of our system be the planar position of the robot and use single integrator dynamics, $\dot{x} = u$, where $u, x \in \mathbb{R}^2$. This is not a bad assumption for mobile robots because you can represent the kinematics of a two-wheeled mobile robot as such after you use an approximate diffeomorphism and feedback linearization [11]. This allows us to easily incorporate navigation behaviors into our dynamics as we let

$$\dot{x}(t) = u(t) = \sum_{i=1}^{m} \gamma_i \beta_i(x(t), O(t)), \qquad (1)$$

where m is the number of behavior functions, $\gamma \in \mathbb{R}^m$ is a vector of behavior weights, $\beta_i(x, O) \in \mathbb{R}^2$ is the i^{th} behavior function, and $O \in \mathbb{R}^{N\times 2}$ is a matrix of the N sensor measurement positions where row j corresponds to o_j^T, the position of the j^{th} sensor measurement.

In [2] the author defines several schema-based behaviors which can be used for navigation; we will use two of these behaviors. However, we must first write them in a form amenable to be used in the equations above, which will allow us to use them in deriving the optimality conditions.

Move-to-Goal: This behavior tries to move the robot towards the goal. Its magnitude is the fixed gain value and the direction is towards the desired goal. We define this as follows.

$$\beta_1(x) = \frac{x_g - x}{\|x_g - x\|} \tag{2}$$

where x_g is the goal state and x is the actual state.

Avoid-Static-Obstacle: This behavior tries to move away from obstacles. Its direction is along the vector pointing from the center of the obstacle to the center of the robot. The obstacle is located a distance d away, it has a sphere of influence S, and the robot is to stay a radius R away from the obstacle. The magnitude is given by

$$\text{mag} = \begin{cases} 0 & d > S \\ \frac{S-d}{S-R} & R < d \leq S \\ \infty & d \leq R \end{cases}$$

To avoid the jump to the infinite input, we will assume that R is small and the minimization of the cost function (which we will design to assign an infinite cost as $d \to 0$) will keep us away from that case. We also sum over all of the obstacles detected by our sensors. We define the avoid-static-obstacle as

$$\beta_2(x, O) = \sum_{i=1}^{N} A(x, o_i) \tag{3}$$

where

$$A(x, o_i) = \begin{cases} 0 & d_i > S \\ \frac{S-d_i}{S-R}\frac{x-o_i}{d_i} & R < d_i \leq S \end{cases}, \; d_i = \|x - o_i\|$$

B. Optimality Conditions

To find the optimal weights for the behaviors we define a cost function that will be a function of the state, $x(s)$, the environment factors (which in this case are the obstacle data measurements), $O(s)$, and the behavior weights, γ. At each time step, we choose the weights that minimize this cost. We write this cost as

$$\int_{t_0}^{t_0+\Delta} L(x(s), \gamma, O(s))ds + g(x(t_0 + \Delta) \tag{4}$$

$$\text{s.t. } \dot{x}(t) = f(x(t), \gamma, O(t)),\ x(t_0) \text{ known}$$

However, we assume that we are in an unknown environment so we do not know $O(s)$ for $s > t_0$, so we simply predict it as $\hat{O}(t_0, s) = O(t_0)$ for $s > t_0$. This will then lead us to estimate $x(s)$ as $\dot{\hat{x}}(t_0, s) = f(\hat{x}(t_0, s), \gamma, O(t_0))$ where $\hat{x}(t_0, t_0) = x(t_0)$. In other words, we are choosing γ solely on our current sensor measurements. This will obviously cause errors in our state trajectory estimation, but will provide a good example for the utility of adapting the time horizon. This allows us to write a well-posed cost function as

$$\begin{aligned} J(\gamma) = &\int_{t_0}^{t_0+\Delta} L(\hat{x}(t_0, s), \gamma, O(t_0))ds \\ &+ g(\hat{x}(t_0, t_0 + \Delta)) \end{aligned} \tag{5}$$

We can now augment (5) with the dynamics to obtain

$$\begin{aligned} \bar{J}(\gamma) = \int_{t_0}^{T} \Big(&L(\hat{x}(s), \gamma, O(t_0)) \\ &+\lambda^T \big(f(x, \gamma) - \dot{\hat{x}}\big)\Big)ds \\ &+g(\hat{x}(T)) \end{aligned} \tag{6}$$

where $T = t_0 + \Delta$ and $\hat{x}(s) = \hat{x}(t_0, s)$ for ease of notation.

Using as standard variational argument, we perturb $\gamma \to \gamma + \epsilon v$ which then causes $\hat{x}(t) \to \hat{x}(t) + \epsilon\eta(t)$. By letting $\bar{J}_\epsilon$ denote $\bar{J}(\gamma + \epsilon v)$ we obtain

$$\begin{aligned} \frac{\bar{J}_\epsilon - \bar{J}}{\epsilon} = \int_{t_0}^{T} \Big(&\big(\frac{\partial L}{\partial x}(\hat{x}(t), \gamma, O(t_0)) \\ &+ \lambda^T(t)\frac{\partial f}{\partial x}(\hat{x}(t), \gamma, O(t_0)) + \dot{\lambda}(t)\big)\eta(t) \end{aligned} \tag{7}$$

$$\begin{aligned} &+ \big(\frac{\partial L}{\partial \gamma}(\hat{x}(t), \gamma, O(t_0)) \\ &+ \lambda^T\frac{\partial f}{\partial \gamma}(\hat{x}(t), \gamma, O(t_0))\big)v\Big)dt \\ &+ \big(\frac{\partial g}{\partial x}(\hat{x}(T)) - \lambda^T(T)\big)\eta(T) + \lambda^T(0)\eta(0) \end{aligned} \tag{8}$$

This gives us the optimality condition

$$\frac{\partial \bar{J}}{\partial \gamma}(\gamma) = \xi^T(t_0) \tag{9}$$

where the costates λ and ξ satisfy the following (backwards) differential equations:

$$\dot{\lambda}(t) = -\frac{\partial L}{\partial x}^T - \frac{\partial f}{\partial x}^T \lambda(t),\ \lambda(T) = \frac{\partial g^T}{\partial x}(\hat{x}(T)) \tag{10}$$

$$\dot{\xi}(t) = -\frac{\partial L}{\partial \gamma}^T - \frac{\partial f}{\partial \gamma}\lambda(t),\ \xi(T) = 0 \tag{11}$$

From (1) we can write our predicted state dynamics as

$$\dot{\hat{x}}(t_0,t) = \beta(\hat{x}(t_0,t), O(t_0))\gamma$$

$$\hat{x}(t_0,t_0) = x(t_0)$$

$$\text{where } \beta(x,O) = \begin{bmatrix} \beta_1^T(x,O) \\ \vdots \\ \beta_m^T(x,O) \end{bmatrix} \in \mathbb{R}^{m\times 2}$$

This allows us to write the partial derivatives of the dynamics in a simple form

$$\frac{\partial f}{\partial \gamma}(\hat{x}(t),\gamma,O(t_0)) = \beta^T(\hat{x}(t),O(t_0))$$

$$\begin{aligned}\frac{\partial f}{\partial x}(\hat{x}(t),\gamma,O(t_0)) &= \sum_i^m \gamma_i \frac{\partial \beta_i}{\partial x}(\hat{x}(t),O(t_0)) \\ &\equiv Q(\hat{x}(t),O(t_0))\end{aligned}$$

We define our instantaneous cost, L, as a cost on control effort and the proximity of the sensor measurements and let it take the form

$$L(\hat{x}(t),\gamma,O(t_0)) = \rho_1 \sum_{i=1}^{N} r(\hat{x}(s),o_i(t_0)) \tag{12}$$

$$+ \frac{\rho_2}{2} u^T(s) R u(s) \tag{13}$$

$$= \rho_1 \sum_{i=1}^{N} r(\hat{x}(s),o_i(t_0)) \tag{14}$$

$$+\frac{\rho_2}{2}\gamma^T\beta(\hat{x}(s),O(t_0))R\beta^T(\hat{x}(s),O(t_0))\gamma$$

where ρ_1,ρ_2, and R are weights and $r(x,o)$ is a repulsive function taking the form $r(x,o) = \frac{1}{2\|x-o\|^2}$. The partials of L required in (10) and (11) can be written as

$$\frac{\partial L}{\partial x} = \rho_1 \sum_{i=1}^{N} \frac{\partial r}{\partial x}(x,o_i) + \rho_2\gamma^T\beta(x,O)RQ(x,O) \tag{15}$$

$$= -\rho_1 \sum_{i=1}^{N} \frac{(x-o_i)^T}{\|x-o_i\|^4} + \rho_2\gamma^T\beta(x,O)RQ(x,O)$$

$$\frac{\partial L}{\partial \gamma} = \rho_2\gamma^T\beta(x,O)R\beta^T(x,O) \tag{16}$$

Finally, to attract the state to the goal, we write the terminal cost as a cost penalizing the distance to the goal.

$$g(x(T)) = \frac{\rho_3}{2}\|x(T) - x_g\|^2 \tag{17}$$

III. Adaptive Look-Ahead

As stated in the introduction, to further improve the performance of our receding horizon approach we would like to find the best time horizon possible at each step. Ideally we would like to look at how well our current prediction will compare with future values, but since this is a noncausal problem we must formulate a causal approximation. Since each proposed solution is only an approximation, we propose two solutions and evaluate their performance through an example.

To do this we make the philosophic assumption that our past ability to predict the state will reflect on our future ability to predict the state. While this is not always the case, it allows us to formulate the problem in a causal manner. If we had a perfect prediction, we would like to make the time horizon as large as possible to improve the underlying optimal control solution. However, we scale back our time horizon when the predictions are poor due to the detrimental effects of a poor prediction.

In [9] we addressed the problem of adjusting the time horizon based on the ability to predict a reference input. However, by looking at the state we are proposing a more general solution and one that is more applicable to robot navigation. This is significantly different than that in [9] due to the fact that errors in the prediction of the state trajectory will not only incorporate errors in estimating reference input but will also include modelling errors and errors resulting from predicting environmental factors such as obstacle positions.

To formulate quality measures that encapsulate the trade-off between the quality of the prediction and the quality of the optimal solution we introduce cost functions with two parts. Each cost function has a point cost, $G(\Delta)$, on the value of the time horizon, Δ, which should be designed to penalize small time horizon values (ie $\frac{1}{\Delta}$). Each cost function also has an instantaneous cost, $F(x,\hat{x})$, that penalizes differences in estimated and actual state values (ie $\frac{\rho}{2}\|x-\hat{x}\|^2$) causing the time horizon to be smaller. This exchanges the tuning of a specific time horizon, which would be very problem specific, for tuning gains that are specific to the performance of the predictor.

A. Looking at the Past

The first quality measure that we propose looks at the past predictions to see how well we have been doing at predicting where the state trajectory will evolve. This is done by summing the contribution of the instantaneous cost between the actual state values and the prediction. We denote our prediction of the actual state, $x(s)$, as $\hat{x}(\tau,s)$ where τ is the time at which the prediction was

made. We let the cost take the form

$$J_{past}(t,\Delta) = \int_{t-\Delta}^{t} F(x(s),\hat{x}(t-\Delta,s))ds + G(\Delta) \tag{18}$$

where

$$\dot{\hat{x}} = f(\hat{x}(t-\Delta,s),u(s))$$
$$\hat{x}(t-\Delta,t-\Delta) = x(t-\Delta)$$

We will solve for Δ using gradient descent strategies. As such, the gradient for J_{past} can be expressed as

$$\frac{\partial J_{past}}{\partial \Delta}(\Delta) = F(x(t-\Delta),\hat{x}(t-\Delta,t-\Delta)) + \frac{\partial G}{\partial \Delta}(\Delta) + p(t-\Delta) \tag{19}$$

Where $p(t) = 0$ and

$$\frac{dp}{dq}(q) = -\frac{\partial F}{\partial \hat{x}}(x(q),\hat{x}(t-\Delta,q))\frac{\partial \hat{x}}{\partial \Delta}(t-\Delta,q) \tag{20}$$

The difficulty with this equation lies in calculating $\frac{\partial \hat{x}}{\partial \Delta}(t-\Delta,q)$ which would be problem dependent or could be estimated by

$$\frac{\partial \hat{x}}{\partial \Delta}(t-\Delta,q) \approx \frac{\hat{x}(t-\Delta-\epsilon,q) - \hat{x}(t-\Delta,q)}{\epsilon} \tag{21}$$

for small ϵ.

B. Testing the Present

The second quality measure looks to see how far back in time our current parameters capture the previous values of the state trajectory. In this cost function we basically run the dynamics backward in time to see how well the current parameters conform to what we have already done. We write the cost as

$$J_{present}(t,\Delta) = -\int_{t}^{t-\Delta} F(x(s),\hat{x}(t,s))ds + G(\Delta), \tag{22}$$

where

$$\dot{\hat{x}} = f(\hat{x}(t,s),u(s))$$
$$\hat{x}(t,t) = x(t)$$

The gradient of $J_{present}$ is much more simple than that of (18) due to the fact that the integrand does not depend on Δ. It takes the form

$$\frac{\partial J_{present}}{\partial \Delta}(\Delta) = F(x(t-\Delta),\hat{x}(t,t-\Delta)) + \frac{\partial G}{\partial \Delta}(\Delta). \tag{23}$$

The main difference between equations (18) and (22) is the fact that the first is testing the prediction using previous parameters and the later is testing predictions using current parameters. Neither one is fundamentally correct as the nature of the problem is noncausal and these are only causal approximations. We will show in section IV that both can provide better outcomes than a constant time horizon.

IV. Navigation Example

In this section we present a simulated navigation example to demonstrate both the utility of adapting the time horizon, as defined in Section III, as well as using receding horizon control to adapt the weights on navigation schema-based behaviors, as defined in Section II.

We ran 2 sets of simulations to verify the usefulness of the proposed adaptation methods. The first set of simulations were setup to simulate a very cluttered environment where the paths could differ significantly while the second was setup to simulate a structured environment where the paths would be quite similar. The resulting paths can be seen in Figure 2.

Each set of simulations included a simulation using constant weights and horizon, constant horizon, and variable horizons using $J_{present}$ and J_{past} where $F(x,\hat{x}) = \frac{\rho}{2}\|x-\hat{x}\|^2$ and $G(\Delta) = \frac{1}{\Delta}$. To allow for a fair comparison, we iterated to determine the best possible weights in each scenario. We also allowed for an initialization period when adapting the time horizons to allow a history of values to be created.

For a comparison between the different methods we evaluated the cost found in (5) with the exception that the limits of integration were from the initial time to the final time. The results for each simulation can be seen in Table I.

In Table I it is evident that the adaptive time horizons performed better than constant time horizons. Also, we observed, for the most part, that the adaptive time horizons were smaller than the best constant time horizons. While we did not evaluate the computation time due to the fact that the dynamics were simple and the computation time was not significant, this could provide an added benefit in the required computation as the system dynamics become more complex.

V. Conclusion

In this paper we have presented a receding horizon method for coordination of schema-based behaviors. We were only able to find two sets of constant weights that would successfully navigate through the structured environment where each of the other methods were quite robust to parameter changes. In both environments, the constant behavior weights had much higher costs than the other methods. This leaves us to conclude that

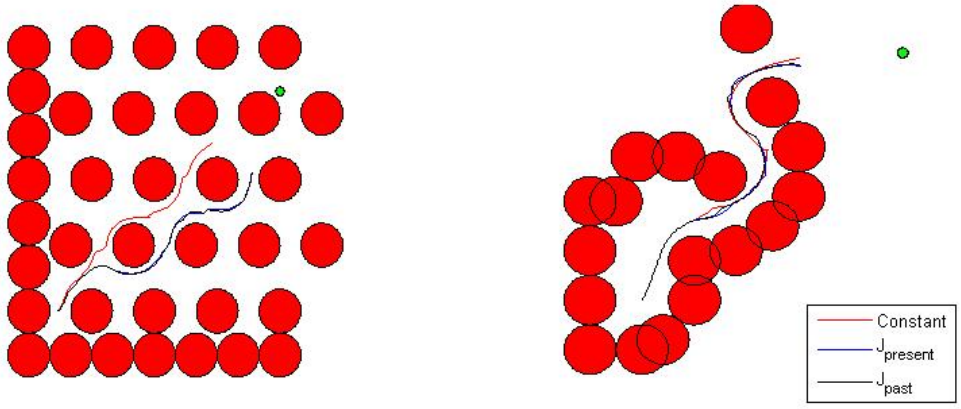

Fig. 2: This figure shows the trajectories through both the cluttered (left) and the structured (right) environments. In each image is shown the trajectory using a constant time horizon, and adaptive time horizon using $J_{present}$, and an adaptive time horizon using J_{past}. The target location is marked with a green circle

	Cluttered Environment		Structured Environment	
	Parameter	Cost	Parameter	Cost
Constant Weights	$\gamma_1 = 40$ $\gamma_2 = 6$	575.4	$\gamma_1 = 39$ $\gamma_2 = 5$	685.1
Constant Horizon	$T = 0.24$	498.1	$T = 0.5$	532.4
Adaptive J_{prev}	$\rho = 100$	479.0	$\rho = 8.4$	521.6
Adaptive $J_{present}$	$\rho = 23.4$	476.6	$\rho = 15.31$	493.23

TABLE I: This table shows the costs and parameters used to calculate the costs for the two sets of simulations.

variable behavior weights using a receding horizon approach can significantly outperform the constant weights method.

We have also present a technique for adapting the look-ahead horizon based upon the quality of the prediction of our state trajectory. In Table I we see that using $J_{present}$ as a quality measure produced better results than using J_{past}. By comparing (19) and (23) we can also see that the gradient for $J_{present}$ is also much easier to evaluate. With these two facts, we believe that $J_{present}$ is a better choice than J_{past}, although a more rigorous analysis is needed to say this conclusively.

Putting both methods together we saw that the that the adaptive time horizons were able to outperformed the constant time horizon in both simulations. This shows the important fact that a variable time horizon can indeed perform better than a constant horizon.

REFERENCES

[1] LaValle, S. Michael. *Planning algorithms*. Cambridge: Cambridge University Press, 2006.

[2] Arkin, R. C.. *Behavior-based robotics*. Cambridge, Mass.: MIT Press, 1998.

[3] Kirk, D. E.. *Optimal control theory: an introduction*. Mineola, N.Y.: Dover Publications, 2004.

[4] D. Q. Mayne, J. B. Rawlings, C. V. Rao, P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality", *Automatica*, Volume 36, Issue 6, June 2000, Pages 789-814.

[5] Ling, K.V.; Yue, S.P.; Maciejowski, J.M.; , "A FPGA implementation of model predictive control," *American Control Conference*, 2006 , vol., no., pp.6 pp., 14-16 June 2006

[6] Camacho, E. F, and C Bordons. *Model Predictive Control, second edition*. London: Springer-Verlag, 2004.

[7] Ogren, P.; Leonard, N.E.; "A convergent dynamic window approach to obstacle avoidance," *Robotics, IEEE Transactions on* , vol.21, no.2, pp. 188- 195, April 2005

[8] T.M. Howard, C. Green, and A. Kelly. "Receding horizon model-predictive control for mobile robot navigation of intricate paths." *Proceedings of the 7th International Conference on Field and Service Robotics*, July 2009.

[9] G. Droge, M. Egerstedt. "Adaptive Time Horizon Optimization in Model Predictive Control". *American Control Conference*, 2011 (Under review).

[10] Rimon, E.; Koditschek, D.E.; , "Exact robot navigation using artificial potential functions," *Robotics and Automation, IEEE Transactions on* , vol.8, no.5, pp.501-518, Oct 1992

[11] Olfati-Saber, R. and R.M Murray. "Near identity diffeomorphisms and exponential epsilon-tracking and epsilon-stabilization of first-order nonholonomic se(2) vehicles." *Porceedings of the American Control Conference*, Anchorage, Alaksa 2002.