2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Fast, Dynamic Trajectory Planning for a Dynamically Stable Mobile Robot

Michael Shomin[1] and Ralph Hollis[2]

Abstract— **This work presents a method to generate dynamically feasible trajectories for a balancing robot in the presence of obstacles, both static and moving. Intended for use on a ballbot, these trajectories respect the dynamics of the robot, and can be generated in milliseconds. Trajectories were experimentally verified on the ballbot in unstructured indoor environments at speeds up to .7 m/s and distances of up to 25 m. The method presented provides a tractable solution for indoor ballbot navigation, enabling safe movement through unstructured environments.**

I. INTRODUCTION

Personal robots need to navigate in dense, cluttered, unstructured environments. The last five years have seen a tremendous amount of research in this area, allowing robots to operate in human environments. This is critical for robots that will eventually provide useful services and interact with humans. Although this capability has been heavily explored for robots such as the PR2 [1] and other statically stable robots, the problem remains unsolved for dynamically stable robots.

In this paper, we present a method for planning collision free paths for the ballbot, a dynamically stable, underactuated robot that balances on a single spherical wheel. We also explore replanning in the presence of dynamic obstacles as well as replanning to account for localization information. Together, these capabilities allow the ballbot to navigate cluttered spaces over long distances in the presence of dynamic obstacles.

Planning for dynamic robots generally requires a higher dimensional plan than kinematic robots, as the dynamics of the robot must be respected in the trajectory planning. Kinodynamic planning [2] tries to solve the problem of concurrent dynamic and kinematic plans. This still requires a high dimensional search space, and as such takes considerable computational effort when attempted with techniques such as graph search. Most efforts attempt to sidestep this bottleneck by using techniques such as RRT* [3]. Such efforts rely on controllable linear dynamics, however, and still need to search a high dimensional space, often making them infeasible for real-time operation.

Achieving dynamically feasible trajectories is critical for the ballbot. Although the robot is 1.75 m tall and weighs 60 kg, dynamically balancing affords the robot inherent compliance, requiring only 3 N of force to be moved. Even though the ballbot is as tall, it is very slender with a body diameter of .4 m. Lastly, the robot is omnidirectional.

*This work was supported by NSF Grant IIS-11165334.

[1]M. Shomin is a PhD Candidate at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15213, USA mshomin at cmu.edu

[2]R. Hollis is a Research Professor, also at the Robotics Institute rhollis at cs.cmu.edu

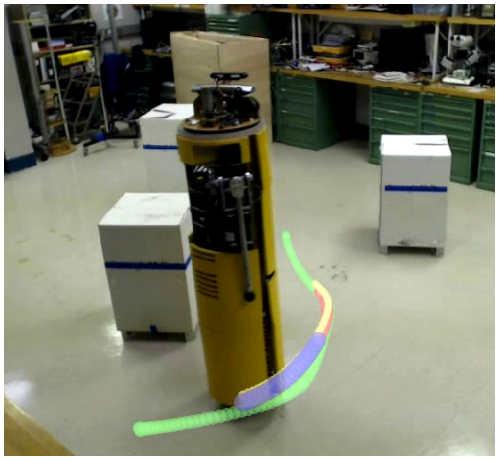


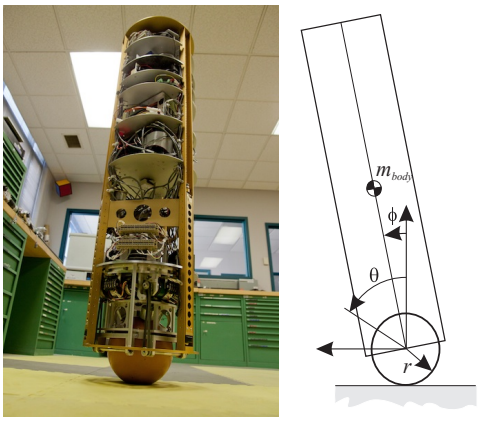
Fig. 1. The ballbot shown executing a trajectory amongst static obstacles (white boxes). The desired path is shown in green. The blue and yellow is a replanned path to account for localization updates, as discussed in Section IV-B.

Previous work on trajectory planning for the ballbot [4] takes previously generated motion primitives [5] that satisfy the dynamics of the robot and sequentially compose them to create longer trajectories. This method yields feasible trajectories for a ballbot but has some drawbacks. The motion primitives are instantiated on a regular grid which means that initial and final configurations are limited to a discrete set of poses. Also, this navigation is not real-time computationally feasible for spaces larger than 5 m $\times$ 5 m. In an effort to solve some of these problems, our more recent work formulated the ballbot as a differentially flat system [6]. Differential Flatness [7] is a property of some dynamic systems which can reduce the complexity of generating feasible trajectories [8]. This prior work was able to generate single point-to-point robot trajectories in milliseconds and execute them, however, multi waypoint trajectories were not investigated.

II. DYNAMIC MODEL AND NOTATION

As in our prior work [6], we consider the ballbot as a decoupled, planar, wheeled inverted pendulum, as shown in Fig. 2, where ϕ is the lean angle, θ the ball angle, l the distance from the center of the ball to the center of mass, radius of the ball r, mass of the sphere m_s, moment of inertia of the sphere I_s, mass of the body m_b, moment of inertia of the body I_b, and gravitational constant g. With these parameters, the equations of motion for the system become:

$$M\ddot{q} + C\dot{q} + G = U, \tag{1}$$



(a) (b)

Fig. 2. Ballbot: (a) balancing on its single spherical wheel, (b) planar ballbot model notation diagram.

$$q = \begin{bmatrix}\theta \\ \phi\end{bmatrix}, M = \begin{bmatrix} I_s + m_b r^2 + m_s r^2 & -m_b l r \cos\phi \\ -m_b l r \cos\phi & m_b l^2 + I_b \end{bmatrix},$$

$$C = \begin{bmatrix}0 & m_b \dot{\phi} l r \sin\phi \\ 0 & 0\end{bmatrix}, G = \begin{bmatrix}0 \\ -m_b g l \sin\phi\end{bmatrix}, U = \begin{bmatrix}\tau \\ -\tau\end{bmatrix}.$$

These two equations can be reformulated into the internal and external system dynamics [5]:

$$M'(q)\ddot{q} + C'(q,\dot{q}) + G'(q) = \begin{bmatrix}\tau \\ 0\end{bmatrix}, \tag{2}$$

where the second equation, the internal system dynamics, is equal to zero on the right hand side.

III. TRAJECTORY GENERATION

A. Differential Flatness

Differential Flatness [7] is essentially a model reduction, where a system is reduced from its full state to its "flat outputs." As long as the flat outputs satisfy appropriate smoothness conditions, the outputs and their derivatives can be mapped back to the full state. In the case of the ballbot, the second equation of motion from (2) can be written as:

$$0 = (\alpha + \beta)\ddot{\theta} + (\alpha + \gamma + 2\beta)\ddot{\phi} - \beta\phi\dot{\phi}^2 + \frac{\beta g \phi}{r}, \tag{3}$$

with $\alpha = I_{\text{ball}} + (m_{\text{ball}} + m_{body})r^2$, $\beta = m_{\text{body}} r l$, $\gamma = I_{\text{body}} + m_{\text{body}} l^2$, each constants. After using a small angle approximation for ϕ, this equation can be used to find a flat output variable S for the system [6]:

$$S = \lambda_1 \theta + \lambda_2 \phi, \tag{4}$$

where $\lambda_1 = r(\frac{\alpha}{\beta} + 1)$ and $\lambda_2 = \frac{r}{\beta}(\alpha + \gamma + 2\beta)$, both constants. As show in prior work [6], this mapping gives the ability to generate a trajectory in the flat output space that satisfies boundary conditions in the original state space. This yields a trajectory that satisfies the equations of motion and is therefore dynamically feasible.

B. Minimum Crackle Formulation

Similar to the generation of trajectories for quadrotors [9], a polynomial basis set can be used in the flat output space. Polynomials are chosen as they satisfy the smoothness constraints of the differential flatness requirements and can be formulated from boundary conditions relatively easily. To generate multi-waypoint trajectories, polynomial segments are put together and constrained to be continuous at intermediate waypoints. It is beneficial to leave the velocity and other derivatives unspecified at these intermediate waypoints, but instead to optimize a trajectory across all waypoints. As seen in quadrotor trajectory generation [9], smooth feasible trajectories can be found by minimizing the snap (fourth derivative of position) over the trajectory, because the quadrotor is a third order system. The ballbot is a 4th order system in flat output space and can exploit the same methods to minimize crackle, the fifth derivative of position. This is equivalent to minimizing the lean angular acceleration, which is proportional to the rate of change of the control input. To minimize crackle, a single polynomial trajectory is first formulated as

$$s(t) = \sum_{i=1}^{9} c_i t^i, \tag{5}$$

where c_i are the coefficients of the polynomial and t is time in seconds. This can be considered as a vector operation:

$$s(t) = c^T p(t), \tag{6}$$

where c is the column vector of coefficients c_i and $p(t)$ is the column vector of t^0 through t^9 evaluated at t. For the convenience of this derivation, the order of c is the c_9 down to c_0. Now let $p_i(t)$ be the *ith* derivative of $p(t)$. Since we are interested in the 5th derivative (crackle) of $p(t)$:

$$p_5(t) = \begin{bmatrix}\frac{9!}{4!}t^4 & \frac{8!}{3!}t^3 & \frac{7!}{2!}t^2 & 6!t & 5! & 0_{1\times 5}\end{bmatrix}^T \tag{7}$$

where $0_{1\times 5}$ is a 1 by 5 vector of zeros. With this formulation, the problem can be posed as the minimization of the sum squared crackle from t_0 to t_f, the start and end times of the polynomial:

$$\min \int_{t_0}^{t_f} (c^T p_5(t))^2 \mathrm{d}t. \tag{8}$$

Now let a be just the nonzero coefficient from (7), and let $\bar{c}$ be the associated polynomial coefficients: c_9 - c_5. Then the objection function becomes:

$$\bar{c}^T \mathrm{diag}(a) \begin{bmatrix} t^8 & t^7 & t^6 & t^5 & t^4 \\ t^7 & t^6 & t^5 & t^4 & t^3 \\ t^6 & t^5 & t^4 & t^3 & t^2 \\ t^5 & t^4 & t^3 & t^2 & t \\ t^4 & t^3 & t^2 & t & 1 \end{bmatrix} \mathrm{diag}(a)\bar{c}. \tag{9}$$

After integration and evaluation from t_0 to t_f, the center matrix of t's becomes:

$$\begin{bmatrix} t_e(9) & t_e(8) & t_e(7) & t_e(6) & t_e(5) \\ t_e(8) & t_e(7) & t_e(6) & t_e(5) & t_e(4) \\ t_e(7) & t_e(6) & t_e(5) & t_e(4) & t_e(3) \\ t_e(6) & t_e(5) & t_e(4) & t_e(3) & t_e(2) \\ t_e(5) & t_e(4) & t_e(3) & t_e(2) & t_e(1) \end{bmatrix} = T, \tag{10}$$

where $t_e(i) = \frac{1}{i}(t_f^i - t_0^i)$. Now, let $H = \text{diag}(a)T\text{diag}(a)$ and it can be seen that integral of sum squared crackle, (8), can be written as

$$\int_{t_0}^{t_f} (c^T p_5(t))^2 \mathrm{d}t = \bar{c}^T H \bar{c}. \tag{11}$$

(11) is now in the form of a cost function for a quadratic program (QP) that minimizes the squared crackle over a trajectory. To use the whole vector c, the cost matrix needs to have zero padding:

$$Q = \begin{bmatrix} H & 0_{5\times5} \\ 0_{5\times5} & 0_{5\times5} \end{bmatrix}. \tag{12}$$

Now the optimization becomes:

$$\min \quad c^T Q c \quad \text{s.t.} \quad Ac = b. \tag{13}$$

Here the equality constraint is the position of the desired waypoints and any fixed derivatives. This formulation extends to multi-waypoint trajectories naturally by concatenating the polynomial coefficients in c of multiple consecutive segments. Q is then repeated along the diagonal to create a block diagonal matrix [9]. In this case, the equality constraints must also enforce consistency, meaning that the derivatives at the end of one polynomial segment must be equal to derivatives at the start of the next segment. This formulation will produce multi-waypoint minimum crackle trajectories, but it is numerically unstable for large trajectories. This is because the decision variables, the polynomial coefficients, can be of vastly different magnitudes. This failure can be seen in Fig. 3(b). The figure shows that although the first quarter of the trajectory is optimized properly, eventually the solver can no longer satisfy the consistency constraints and the path is discontinuous. Fortunately, the problem can be formulated as a much more stable, unconstrained QP.

C. Unconstrained Optimization

The key insight to reformulating the optimization is that the problem can be formed as an equivalent QP, but with the waypoint positions and derivatives as the decision variables instead of the polynomial coefficients. Constructing the problem this way both removes the equality constraints and uses decision variables bounded to a much smaller range. This change of variable has actually already been computed in (13). The A matrix in this equality constraint converts the coefficients c to the positions and derivatives, b. However, these are expressed in terms of the flat outputs, and it is more desirable to express these parameters in terms of the state d. Using d as the decision variable, the cost function can be reformulated as the total cost J:

$$J = d^T C A^{-T} Q A^{-1} C^T d, \tag{14}$$

where C is a selector matrix of ones and zeros which can rearrange the order the decision variables in the vector d. This derivation of this cost function is identical to its original derivation for quadrotors [10]. The key difference is that quadrotors are trivially differentially flat, in that their state variables are the flat outputs. This is why the transformation matrix C can be only zeros and ones. For ballbot, or any other nontrivially flat system, another transformation matrix is required to go from polynomial coefficients to state variables. From (4), the necessary transformation M can be found as:

$$Ac_s = b, Ac_s = Md \tag{15}$$

$$c_s = A^{-1}Md \Rightarrow d^T M^T A^{-T} = c_s^T. \tag{16}$$

where M is a block diagonal matrix with matrices m along the diagonal:

$$m = \begin{bmatrix} \lambda_1 & 0 & \lambda_2 & 0 & 0 \\ 0 & \lambda_1 & 0 & \lambda_2 & 0 \\ 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & g \end{bmatrix}. \tag{17}$$

M is now a matrix that transforms flat outputs and derivatives: $b = [S\dot{S}\ddot{S}S^{(3)}S^{(3)}]^T$ to state variables: $d = [\theta\dot{\theta}\phi\dot{\phi}\ddot{\phi}]^T$. It is worth noting that A in (15) and its inverse can be solved in closed form as a function of the segment time. With this new transformation M, (14) becomes

$$J = d^T R d, \tag{18}$$

where R is

$$R = C M^T A^{-T} Q A^{-1} M C^T. \tag{19}$$

This matrix can be partitioned into fixed and free derivatives along with the vector of ballbot states d [10]:

$$R = \begin{bmatrix} R_{FF} & R_{FP} \\ R_{PF} & R_{PP} \end{bmatrix}, d = \begin{bmatrix} d_F \\ d_P \end{bmatrix}. \tag{20}$$

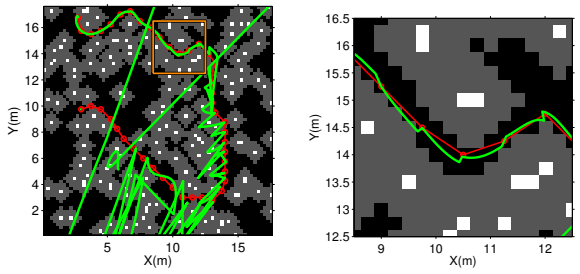
d_f is the vector of fixed derivatives. Formulated as such, the optimal free waypoint derivatives, d_p^*, can be solved for as:

$$d_p^* = -R_{PP}^{-1} R_{FP}^T d_F \tag{21}$$

This unconstrained formulation is both more numerically stable and of lower dimension than the constrained formulation. A comparison to the constrained method can be seen in Fig. 3(c). This trajectory optimizes over 44 waypoints. The first and last waypoints are fully constrained as they are set to be at rest with zero velocity and lean angle. The 42 interior waypoints are constrained only in position, while the other derivatives are free to be optimized. This example was solved in MATLAB using gaussian elimination on an Intel Core I7 machine in 1.1 ms. Conversely, the constrained system in Fig. 3(a) took 76 ms and still diverged. The unconstrained method is also very stable, with trajectories up to 250 waypoints being tested.

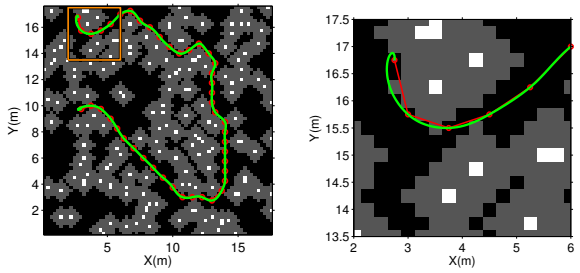
D. Polynomial Segment Time-Allocation Optimization

Minimizing energy of a series of polynomial segments, as shown in the previous section, relies on knowing how much time each segment should take. From (10), it is clear that t_f and t_0 must be known for each polynomial segment before constructing the T matrices. The choice of these segment switching times is very important, as poorly allocated times will impose unwanted constraints on the robot motion, as shown in Fig. 4. In this figure, a trajectory is generated which is constrained to take a total of 5 s and pass through the 5 yellow waypoints, starting and ending at rest. The blue path is the trajectory generated by naively choosing



(a) Interior Point QP Solver solution for the constrained problem

(b) Magnified view of orange box from (a)



(c) Unconstrained QP solved as a linear system

(d) Magnified view of orange box from (c)

Fig. 3. Optimization of the same kinematic path (shown in red) using different techniques. Free space is shown in black, obstacles in white, and obstacle inflation in gray. Because this trajectory has 44 waypoints, the unconstrained optimization shown in (c) and (d) is the only successful method yielding a feasible dynamic trajectory.

equal times, 1.25 s, for each segment. The red path shows a trajectory with times that were optimized by iteratively solving the QP and using gradient descent to yield a locally minimal energy [10]. The green dashed path uses no time optimization, but instead allocates time based on a heuristic. This heuristic calculates times to satisfy the dynamics of a constant acceleration system with a capped maximum velocity. The following algorithm details the time allocation for the polynomial segment times, t_i. The algorith uses v_m, a specified maximum velocity; a, the specified acceleration; d_s the polynomial segment euclidian distance; v_0 the segment's inital velocity; and v_f, the segment's final velocity.

Algorithm 1 Time-Allocation Heuristic

```
Assign waypoint velocites from kinematic plan
for all segments do
    t_1 ⇐ |v_m − v_0|/a
    d_1 ⇐ ((v_0 + v_m)/2) ∗ t_1
    t_2 ⇐ |v_m − v_f|/a
    d_2 ⇐ ((v_f + v_m)/2) ∗ t_1
    if d_1 + d_2 < d_s then
        t_m ⇐ (d_s − (d_1 + d_2))/v_m
        t_i ⇐ t_1 + t_m + t_2
    else
        t_i ⇐ t_1 + t_2
    end if
end for
```

The unoptimized trajectory in Fig. 4 (blue) requires a maximal lean angle of $4.07°$, whereas the optimized trajectory (red) requires only a $1.63°$ maximum lean angle. For the ballbot, this corresponds to an instantaneous acceleration of .82 and .33 m/s^2, respectively. Note also that these trajectories take the same total amount of time. The trajectory which has times allocated by the heuristic has a maximum lean angle of $1.67°$, slightly more than the optimized trajectory. Although the optimized trajectory has lower required acceleration, using the heuristic has two major advantages: it requires no extra optimization step, and it is not susceptible to local minima.

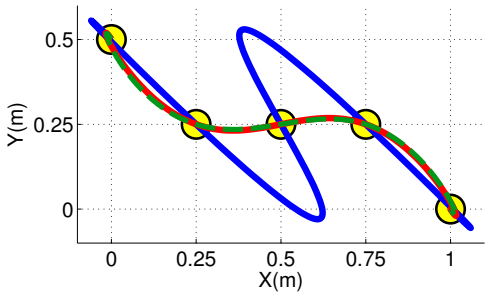


Fig. 4. Comparison of the optimized trajectories using different methods of time allocation. Blue - Equal times (naive), Red - Optimized times, Green Dashed - Heuristic

IV. PLANNING TRAJECTORIES IN THE REAL WORLD

A. Planning with Obstacles

As discussed in the previous section, a feasible trajectory for the ballbot can be generated from a series of waypoints; however, this does not take obstacles into account. To this end, a graph search planner is used to generate a kinematic, obstacle-free trajectory. That trajectory is then subsampled into waypoints to optimize a minimum crackle trajectory which is dynamically feasible by the strategy presented in the previous section. An example of this can be seen in Fig. 3(c) and more closely in Fig. 3(d). The red path is a kinematically feasible two-dimensional path generated by A*, and the green trajectory is dynamically feasible.

This method is very powerful, as the graph search is done in a two dimensional space, instead of the 8 dimensional full state space. Fig. 3(c) shows obstacles as white grid cells, with an inflation shown as gray cells. To ensure that the deviation of the dynamically feasible path from the kinematic path does not hit an obstacle, the obstacles must be inflated by approximately the distance between subsampled waypoints.

B. Replanning with Localization Updates

As will be discussed in Section V-A, the ballbot uses a 30 m laser scanner and Hector Slam [11] as a localization solution. This localization information allows for a much better estimate of the robot state than odometry alone. As such, this estimate or a combination of localization and odometry information are usually used for feedback control, as in prior work with the ballbot [5]. Unfortunately, localization systems can yield discontinuous estimates, especially those based on scanmatching.

This method takes a different approach which can utilize localization information immediately while using only continuous odometry for feedback control. Instead of filtering the localization information, a single polynomial trajectory is planned from the current state as estimated from localization

to the global trajectory at some time in the future. This trajectory is then executed by a feedback controller using only odometry information, which is always continuous. This trajectory is then replanned to take into account updated localization. Generating this intermediate trajectory leverages the ability to compute a single polynomial trajectory in less than a millisecond. Executing feedback control with only odometry information yields another benefit: the planning and control can be decoupled. This is useful in the case of the ballbot because balancing control is done on a real-time computer, while planning and localization are run on a more standard, high-level computer. An example of this method in action can be seen in Fig. 5.

C. Planning Safe Trajectories

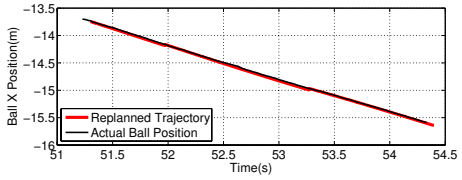
In the case of localization failure or a total high-level system failure, it is desirable for the robot to stay balanced. The previous section introduced the ability to decouple the system in planning and control. This decoupling provides another benefit because if the real-time control system can bring the robot to rest by itself in the event of a high-level failure, the robot is much safer. To this end, the proposed trajectory generation method was designed to only ever send 1.2 s of the desired trajectory to the real time system, followed by another trajectory which brings the system to rest, as in [6]. As long as everything works properly, the high-level replanner sends another 1.2 s trajectory before the previously sent trajectory has been completed. If the high-level planning fails for any reason or generates a plan that is infeasible, the low-level system reverts to the backup trajectory and comes to rest. This value of 1.2 s was chosen empirically trading off safety and computational load. An example of this backup trajectory method can be seen in the accompanying video.

V. RESULTS

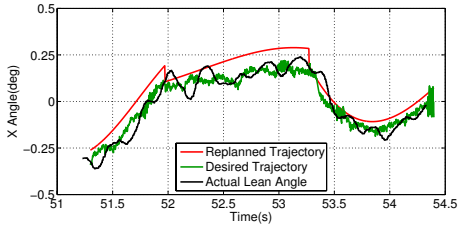
A. Experimental Setup

Trajectories were tested extensively over 60 trials, in a 5 m $\times$ 8 m room in the presence of static and dynamic obstacles. Average velocities were varied from .3 m/s to .7 m/s. Ten experiments tested the stability of generating long trajectories. These trajectories were all over 20 m and traversed rooms and hallways with floors of carpet and tile. The timing of the trajectory segments were allocated using the heuristic method discussed in Section III-D. Plans were checked once per second for continued feasibility. If an obstacle had invalidated the planned trajectory, a new trajectory was generated. New goals were also commanded to the robot while already executing a trajectory to assess the ability to smoothly transition to a new trajectory for a new goal.

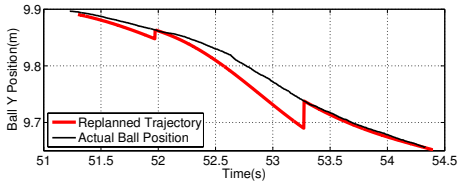
The robot was controlled using an inner loop, modified PID balancer and an outer loop trajectory tracking controller, as in prior work [4]. The control was modified slightly to use feedback on the center of mass position instead of the ball position in the outer loop. This was motivated by the feedback linearization presented by the flat outputs.



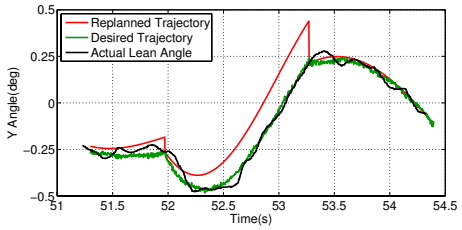
(a) Ball X Position tracking over 2 m in 3 s



(b) X Lean Angle tracking. The planned trajectory is in red, the controller output in green, and the actual angle in black



(c) Y Position tracking .25 ms in 3 s. This portion of the total path was primarily moving in the x direction.



(d) Y Lean Angle tracking. The planned trajectory is in red, the controller output in green, and the actual angle in black

Fig. 5. Planned and actual state data from an experiment with the ballbot. The entire experiment had a duration of 90 s and traversed over 20 m through 2 rooms and a hallway with obstacles. Seconds 51 through 55 are shown to highlight position and lean angle tracking along with 1.2 s period replanning strategy. The replanning that occurs at t = 52 s has relatively poor localization, but the replannned trajectory at 53.2 s returns to the global trajectory very adequately.

B. Experimental Results

The ballbot was able to successfully navigate a building in the presence of dynamic obstacles at speeds up to .7 m/s. Fully dynamic, feasible trajectories up to 25 m long were planned in less than 50 ms. A piece of one such trajectory is shown in Fig. 5, with the full trajectory shown in Fig. 6. This particular trajectory was 90 s long, moving through 2 doorways and a hallway at .6 m/s. Fig. 5 highlights the replanning for localization, as well as the tracking accuracy of the system.

Fig. 5(a) shows the ball position tracking of the trajectory. As this particular segment of the trajectory was mostly in the x direction, over the 3 seconds shown, the ballbot moved almost 2 meters. As such, the difference between the desired trajectory and the actual performance is almost indistinguishable at this scale. Shown in Fig. 5(b) is lean angle in the x

direction. Replanning to account for localization occurred at t = 52 s and 53.2 s. Fig. 5(c) shows the tracking in the y direction. Note the scale, as this figure shows only .25 m of travel. This figure clearly shows that the replanning events use the state of the ballbot, both lean angle and position, as the initial conditions. The replanning event at t = 52 s actually uses a poor localization estimate, off by .05 m. As such, it looks like the ballbot is .05 m from where it should be in the y direction, but cannot compensate by the time the next trajectory is planned. By the time the next replanned trajectory is generated at t = 53.2, the localization has returned to a better estimate, as is clearly seen from the smoothness and good tracking of the next replanned segment. Because this method of trajectory generation is so fast,

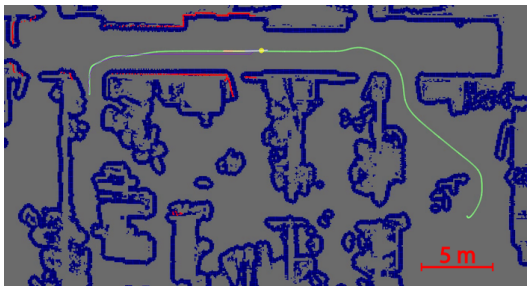
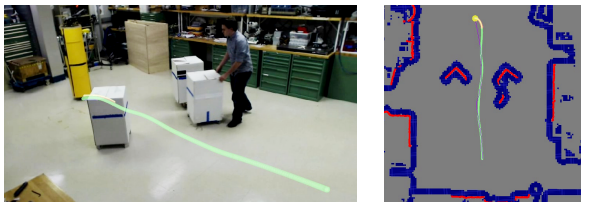


Fig. 6. Top view of ballbot executing a 25 m trajectory through a building. The desired path is shown in green; laser scanner returns are shown as red dots; Inflated obstacles are shown in blue, and the current ball position is shown as a yellow dot.

generating plans in less than 50 ms, it is very appropriate for dynamically replanning in the presence of moving obstacles. This ability is shown in Fig. 7. Fig. 7(a) shows the initial situation, where the ballbot generates a trajectory shown in green. At four seconds into the experiment, a box is placed directly in the path of the robot, and the robot replans a trajectory to its right, as seen in Fig. 7(c). It is important to note that all planned trajectories begin with the ballbot's current state as the initial condition. As such, the robot smoothly transitions from an old trajectory to a new one. Lastly, the second path of the ballbot is blocked by a person, as shown in Fig. 7(e).

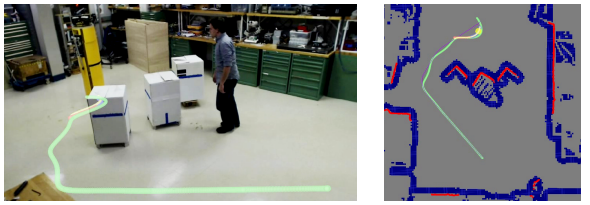
VI. CONCLUSIONS

A method of generating dynamically feasible trajectories for a ballbot in the presence of obstacles has been presented. This method has been shown to produce smooth motions, even over very large distances. The generation of these paths is also sufficiently fast for replanning when faced with moving obstacles, and replanning to account for localization information. Trajectories produced have been experimentally verified on the ballbot at speeds up to .7 m/s. This is a significant speed for a person sized robot, matched by very few other systems. The method presented also always ensures both a smooth trajectory and smooth transitions between segments. Furthermore, to our knowledge, this is first time a method has successfully enabled a ballbot to navigate unstructured environments at such speeds or distances. This advancement provides a real, tractable solution to autonomous ballbot indoor navigation.



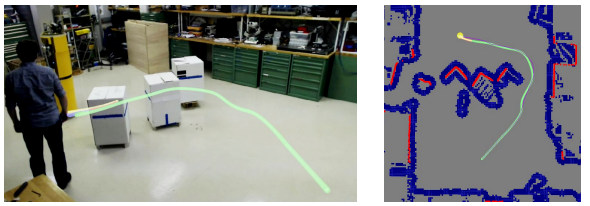
(a) The ballbot initially starting at rest at $t_0 = 0$ s. A straight path to the goal in the lower right is planned.

(b) Overhead view of (a)



(c) At $t_1 = 4$ s, a box is placed blocking the straight line path, and the ballbot replans smoothly to go around the obstacles.

(d) Overhead view of (c)



(e) At $t_2 = 11$ s, a person block the path of the ballbot again

(f) Overhead view of (e)

Fig. 7. The ballbot navigates a room in the presence of dynamic obstacles. The planned trajectory is shown in green; laser scan, red; Inflated obstacles, blue, and the current ball position, yellow

REFERENCES

[1] E. Marder-Eppstein, E. Berger, T. Foote, B. P. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *International Conference on Robotics and Automation*, 05/2010 2010.

[2] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *J. ACM*, vol. 40, no. 5, pp. 1048–1066, Nov. 1993.

[3] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 5054–5061.

[4] U. Nagarajan, B. Kim, and R. Hollis, "Planning in high-dimensional shape space for a single-wheeled balancing mobile robot with arms," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 130–135.

[5] U. Nagarajan, G. Kantor, and R. Hollis, "Trajectory planning and control of an underactuated dynamically stable single spherical wheeled mobile robot," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, may 2009, pp. 3743 –3748.

[6] M. Shomin and R. Hollis, "Differentially flat trajectory generation for a dynamically stable mobile robot," in *2013 IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, May 2013, pp. 4452–4457.

[7] M. Fliess, J. Lévine, and P. Rouchon, "Flatness and defect of nonlinear systems: Introductory theory and examples," *International Journal of Control*, vol. 61, pp. 1327–1361, 1995.

[8] H. Sira-Ramírez and S. Agrawal, *Differentially Flat Systems*, ser. Control Engineering. Taylor & Francis, 2004.

[9] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[10] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *International Symposium on Robotics Research*, Singapore, Dec. 2013.

[11] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.