Modelling Computational Requirements of Mobile Robotic Systems Using Zones and Processing Windows
TR-UNL-CSE-2007-0016

Ala′ Qadi Steve Goddard
Computer Science & Engineering
University of Nebraska–Lincoln
Lincoln, NE 68588-0115
{*aqadi, goddard*}*@cse.unl.edu*

Jiangyang Huang
Itron, Inc
West Union, SC 29696
jiangyang.huang@itron.com

Shane Farritor
Mechanical Engineering
University of Nebraska–Lincoln
Lincoln, NE 68588-0656
sfarritor@unl.edu

Abstract

Mobile robotic systems must sense constraints imposed by a dynamically changing environment and predictably react to those changes in real-time. Complexity arises in mobile robotic systems because the computing platform travels through the environment with which the system is interacting. These systems have spatio-temporal requirements in the sense that correct behavior is defined in terms of both space and time. The focus of this paper is mobile robotic platforms that must sense their environment and avoid obstacles as they navigate from one point to another. We present a design and analysis methodology for these platforms that integrates spatio-temporal attributes with fixed priority real-time scheduling through the use of zone and processing window abstractions.

1. Introduction

Mobile robotic systems must sense constraints imposed by a dynamically changing environment and predictably react to those changes in real-time. Mobile robotic systems add yet one more level of complexity in that the computing platform travels through the environment with which the system is interacting. These systems have spatio-temporal requirements in the sense that correct behavior is defined in terms of both space and time. As real-time systems, computations must be completed within established response times, but they may also have varying temporal requirements. As spatial systems, the computations performed and their timeliness will be dependent on (i) the location of the platform in its environment, (ii) the velocity with which the platform is moving, and (iii) the existence of objects in the environment.

We present a design and analysis methodology for these platforms that integrates spatio-temporal attributes with fixed priority real-time scheduling. To support dynamic environments, we divide the path the mobile platform traverses into zones and associate with each zone a processing window. The spatial dimensions of each zone are dependent on the platform's sensing capabilities and the existence of obstacles in the zone. The processing window represents the time interval required to scan a zone and plan a safe path through that zone. To ensure schedulability, lower bounds for the processing window are derived that account for both the task set and attributes of the sensors needed to navigate. The speed at which the platform can travel is limited by physical attributes of the platform and the minimal feasible processing window (since this limits the sampling rate of sensors). The challenge, however, is that the obstacles in the environment also limit platform speed, minimal processing windows, or both simultaneously. Thus, we present a technique for adjusting platform speed and processing windows such that the maximum speed less than or equal to the desired speed is maintained while adjusting the processing window to maintain schedulability of the platform's real-time tasks.

The main contributions of this paper are as follows:

- an abstract analysis methodology for mobile robotic systems that integrates spatio-temporal attributes with fixed priority real-time scheduling; and
- an algorithm for adjusting platform speed and processing windows such that the maximum speed less than or equal to the desired speed is maintained or maximized while adjusting the processing window to maintain schedulability of the platform's real-time tasks without upgrading the platform processor.

The remainder of this paper is organized as follows. Section 2 discusses background and related work. Section 3 presents our zone and processing window abstractions. Section 4 integrates the spatio-temporal attributes of zones and processing windows with real-time scheduling. Section 5 shows how the presence of obstacles in the environment affects the maximum platform speed, the processing window, or both. Then an algorithm to automatically make these adjustments is presented. Our design and analysis methodology for mobile robotic systems is evaluated via simulation

in Section 6 and on a real platform in Section 7. Finally, a short conclusion is presented in Section 8.

2. Background and Related Work

This section provides a summary of the background and related work. Section 2.1 provides a summary of background information and Section 2.2 provides a summary of related work.

2.1 Background

The term real-time is often used in the robotics community to signify some sort of reactivity to external events, or a capability to respond to environmental changes [6, 12, 4]. In fact, most robotic systems are intended to be "real fast" as opposed to real-time. The execution time of their system services and internal operations are designed to be as fast as possible in order to minimize the average execution times. Such systems may successfully operate in real-time, and provide a cost-effective solution for certain applications. However, in many uncertain situations, the robotic system might be overloaded and cannot complete some tasks within their deadlines, which can result in an undesired system behavior.

Thus, the software architecture for the overall control of a sophisticated robotic system must have real-time characteristics. In other words, a real-time robotic system is expected to not only work correctly but also respond to external events deterministically. It is desired to have *deterministically* fast response to urgent events. All tasks are required to meet their respective timing requirements. Even when the system is overloaded by unexpected events and meeting all deadlines is impossible, it is expected to guarantee the deadlines of selected critical tasks. Real-time analysis and design can significantly improve the quality of service delivered by the mobile robotic system, and allows the designer to separate the concern of the system's logical correctness from the concern to meet the system's timing constraints (i.e., temporal correctness).

A real-time system is a system that is required to complete its work and deliver its services on a timely basis. The main difference between a real-time system and a normal system is that a real-time system is not just required to produce the correct output, but to produce the correct output on time. Before introducing the periodic real-time model, we present some essential terms used in real-time systems.

Task: A sequential piece of code that is executed repeatedly with some pattern.

Job: An instance of an execution of a task.

Release time of a job: The time instant the job becomes ready to execute.

Deadline of a job: The time instant by which the job must complete execution.

Relative deadline of a job: Deadline minus the release time of the job.

There are two primary types of timing constraints in real-time systems: *hard* and *soft*. In hard real-time systems, a late job is not allowed because it may cause disastrous consequences. Late completion of a job that has a soft deadline is allowed because a few misses of soft deadlines do not produce serious harm.

The most widely used and accepted real-time model is the periodic task model as defined by Liu and Layland [15]. According to the periodic task model, the real-time system is defined by a set of tasks $\mathrm{T} = \{T_1, T_2, \ldots, T_n\}$. Every task releases jobs at a constant interval for each task. The task is defined by four parameters $\phi, p, e,$ and d :

ϕ is the phase of the task—the interval of time between the instant the system started and the release of the first job of the task;

p is the period between the release of two consecutive jobs of a task;

e is the worst-case execution time of the job;

d is the relative deadline for the job.

In real-time systems there are two main schemes used to dynamically schedule jobs online: Fixed Priority Scheduling and Dynamic Priority Scheduling.

- **Fixed Priority Scheduling**: A fixed priority scheduling algorithm assigns the same priority to all jobs of a task. The priority does not change during application execution. The scheduling decision is made based on the task priority. When a task is released, all *jobs* for this task can only be delayed by higher priority tasks or interrupts. A well known fixed priority algorithm is the *rate monotonic priority assignment* (RM) [15]. This algorithm assigns task priorities based on the rate the tasks execute. The task with the highest rate of execution is given the highest priority. The task with lowest rate of execution is given the lowest priority.
- **Deadline driven scheduling**: Deadline driven scheduling is a dynamic scheduling algorithm that assigns priorities to tasks according to the deadline of their current requests. At each time instant, the scheduler assigns a priority to each ready-to-execute job and allocates the highest priority job to the processor. This method of assigning priorities to tasks is a dynamic one and it is different from fixed priority algorithms. One of the well known dynamic scheduling algorithms is the *earliest deadline first* (EDF) scheduling algorithm [15]. In EDF scheduling, at any time instant, it assigns the highest priority to jobs with the earliest absolute deadline. EDF is known to be an *optimal* scheduling algorithm for the class of dynamic scheduling algorithms. For a given set of m tasks with an optimal deadline driven scheduling algorithm, the task set is feasible if and only if

$$U = \sum_{i=1}^{m} e_i/p_i \leq 1.$$

The deadline driven scheduling algorithm is able to achieve full processor utilization.

2.2 Related Work

Applying traditional real-time systems scheduling theory to robotic applications is not a new concept. Examples of applying the real-time scheduling to robotics can be found in [7, 9, 22, 3, 17, 18, 23, 2, 16, 10, 13, 14]. Of these, rate monotonic (RM) [15] scheduling was used in [9, 22, 3, 17]; earliest deadline first (EDF) [15] scheduling was used in [17, 23]; and feedback based scheduling techniques in [22, 10, 14]. The work in [16, 13] used real-time scheduling theory to assign tasks to robots in a team of mobile robots. While these papers applied real-time scheduling theory to a robotic application, none of them considered the execution requirements of the platform's sensing and planning as a factor in the platform's velocity calculations. Moreover, the current literature does not address the issues of using fixed priority scheduling within any form of processing window for mobile platforms whose workload changes with the environment. The remainder of this section provides a brief summary of some of the contributions made by the most significant, or most closely, related work.

The earliest example of applying some form of real-time scheduling to the field of robotics is the seminal work by Dertouzos [7] in which he proved the optimality of EDF in underloaded conditions. It appears that George and Kanayama [9] were the first researchers to apply canonical RM scheduling to an autonomous mobile robotic application.

Wargui et al. [22] used real-time scheduling theory to address communication time delays in the sensing, control and action feedback loops of the control system in a mobile robot. The mobile robot is seen as a system with message queues controlled through a multiplex communication link. The delays are included in the derivation of a scheduling bound for RM scheduling.

Beccari et al. [3] presented rate modulation scheduling techniques for adaptation of soft real-time loads to available computation capacity in the context of autonomous robot control architectures. Their methods are based on the knowledge of worst-case execution time of tasks and are focused on period adjustment of soft real-time tasks within a range of admissible rates.

Prasad and Burns [18] proposed a pre-runtime method for ranking services on an autonomous vehicle system. Their method assigns a value for each service based on many factors including the time the service completes, the history of invocations of the service, importance of the service, and state of the computer system that the services are being run on.

Baccelli et al. [2] used petri nets and marked graphs to analyze the temporal correctness of periodic real-time tasks under preemptive fixed priority scheduling. They applied their work to a specific software environment dedicated to the design, verification and the implementation of robotic control systems (ORCCAD).

Miyata et al. [16] developed a task assignment system for a team of robots handling flexible materials. Their task assignment algorithm used task templates to divide the work done by robots into tasks and assigned the tasks to robots based on the number of free robots and task priorities. However, their work did not consider hard deadlines or real-time scheduling theory. Neither did it relate any of the real-time requirements to the robots's velocity.

Li et al. [13] proposed a method that converts robotic applications into strategies that can be modelled with acyclic task graphs implemented as periodic tasks. They then presented an algorithm to distribute the periodic tasks to a team of mobile robots.

Lin et al. [14] present a feedback-based real-time adaptive scheduling method for an autonomous vehicle that is used to spray herbicide in agriculture production. They used the idea of feedback control to adjust the speed of the vehicle based on the a deadline miss ratio and CPU utilization of the vehicle system.

While many of the previously mentioned papers applied real-time scheduling theory to a robotic application, none of these papers considered the execution requirements of the robots sensing and planning as a factor in the robot's velocity calculations. The current literature does not address the issues of using fixed priority scheduling of processing windows for mobile platforms whose workload changes with the environment.

The closest work to this paper, other than our own, is by Hassan et al. [10]. Their work considers the variability of the system load and temporal requirements. They use a feedback control scheduler (FCS) and a flexible server (FS) for a hybrid mobile robotic system (deliberative and reactive). The FCS scheduler permits the adaptation of the temporal requirements. However, their work does not relate velocity calculations to the robot's sensing abilities or changes in the environment.

In [21] we used real-time scheduling theory to address the challenge of a lead robot controlling the placement of less capable Robotic Safety Markers (RSMs) [21]. We extended the functionality of the RSM system by adding the capability of the RSMs to follow the lead robot in [11]. In [19], we showed how real-time scheduling analysis could be applied to a specific mobile robotic application in which the periods of tasks were dependent on the speed at which the robot was moving. This paper generalizes and extends the technique first applied in [19] by creating the zone and processing window abstractions, and then using those abstractions to perform a more general scheduling analysis.

3 Zones and Processing Windows

A mobile robotic platform collects data from its sensors to build a map of its environment. This data is then com-

bined with mission goals to plan a path to its next destination. The speed and direction of the platform, represented by a velocity vector, is dependent on the number of obstacles in the environment and how soon the platform must reach its destination. To integrate the spatio-temporal attributes of the platform with fixed priority real-time scheduling theory, we have created zone and processing window abstractions, which are presented in Section 3.1. Section 3.2 then presents a technique to correct distance errors that occur due to the asynchronous nature of scanning the environment while the platform is moving through a zone. In the remainder of this paper, the (shorter) term platform will be used to denote a mobile robotic platform.

3.1 The Abstractions

The platform's intended area of exploration is divided into subareas called *zones* so that we can isolate the computational and spatial (speed) requirements for each zone and perform the analysis separately on each zone. We define a zone as the area for which the platform collects and processes sensor information, creates a map for the area and plans its path through the area. Each zone is associated with one desired speed for the platform.

The zone boundary is defined by the region of exploration in which the platform can build a map and safely generate a path trajectory using previously collected sensor information. Figure 1(a) shows an example of the zone boundary where the platform starts collecting sensor data at point A, and does not move while collecting the sensor data. In this case the zone boundary is defined by the sensors' range. In Figure 1(a) the platform uses sensors with an angle of coverage of θ and maximum range of r. In Figure 1(b) the platform starts collecting data at point A and continues to move while collecting sensor data. It is not until point B that the platform is able to build a map for its intended area of exploration based on its sensor information. At this point all sensor readings taken on the platform path from point A to point B must be converted relative to point B. Therefore the zone boundary is reduced. The zone in Figure 1(c) is further reduced because a safety area has been added for extra precautions due to sensor errors and braking distance. Another factor that affects the zone boundary is the existence of objects that limits visibility beyond the object.

In the two dimensional zones shown in Figure 1, the zone is a circular section due to the sensor distribution and coverage. Therefore each zone boundary can be defined by an angle and a radius. From Figure 1(c) we can see that the *zone radius* D_i is equal to the sensor range r minus the distance the platform moved from point A to point B minus the width of the safety stopping distance S_M. Therefore D_i can be calculated from Equation (1), where $\overline{AB}$ is the distance the platform moved from point A to point B.

$$D_i = r - \overline{AB} - S_M \tag{1}$$

In this context, the zone can be any two or three-dimensional shape depending on the distribution and range of sensors on the platform. For ease of demonstration we will only consider two dimensional zones in which all platform sensors provide a two dimensional map. Extension to three dimensions follows the same concepts used in our two dimensional zone model.

We define the point (in space and time) that the platform finishes planning its path and speed for zone Z_i as *planning point* F_i. Because F_i describes both spatial and temporal information, *planning point* F_i is denoted by the tuple (t_i^F, L_i^F) where t_i^F represents the time instant the platform arrives at the point F_i and L_i^F represents the platform position information at point F_i. L_i^F is also a tuple whose parameters depend on the nature of the required position information and the coordinate system used. In our case, our mobile robotic system involves a robot that moves in a two-dimensional cartesian coordinate system. Therefore for the rest of this paper L_i^F will be denoted by the tuple (x_i^F, y_i^F, ψ_i^F) where x_i^F and y_i^F represent the platform's x and y coordinates respectively and ψ_i^F represent the platform's orientation angle. If we expand the L_i^F tuple in the tuple (t_i^F, L_i^F) we can represent planning point F_i in a two dimensional cartesian coordinate system by the four parameter tuple $(t_i^F, x_i^F, y_i^F, \psi_i^F)$.

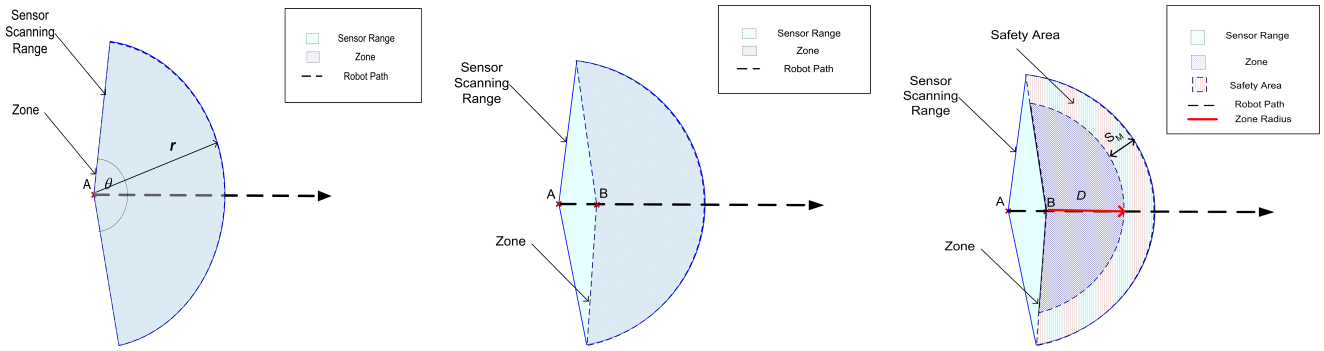
Each zone Z_i is bounded by the two planning points: F_i and F_{i+1}. The platform collects sensor data through zone Z_i. The platform's planing for the next zone Z_{i+1} must be finished by the end of the next planning point, F_{i+1}. Therefore the platform must finish collecting data, build an environment map and plan for the next zone, Z_{i+1}, before the platform starts moving through zone Z_{i+1}.

We define the *zone processing window* W as the time interval from the instant the platform starts collecting data to the moment the platform finishes planning for the zone. Therefore $W_i = [t_i^F, t_{i+1}^F)$, and the processing window duration w can be calculated from Equation (2).

$$w_i = t_{i+1}^F - t_i^F \tag{2}$$

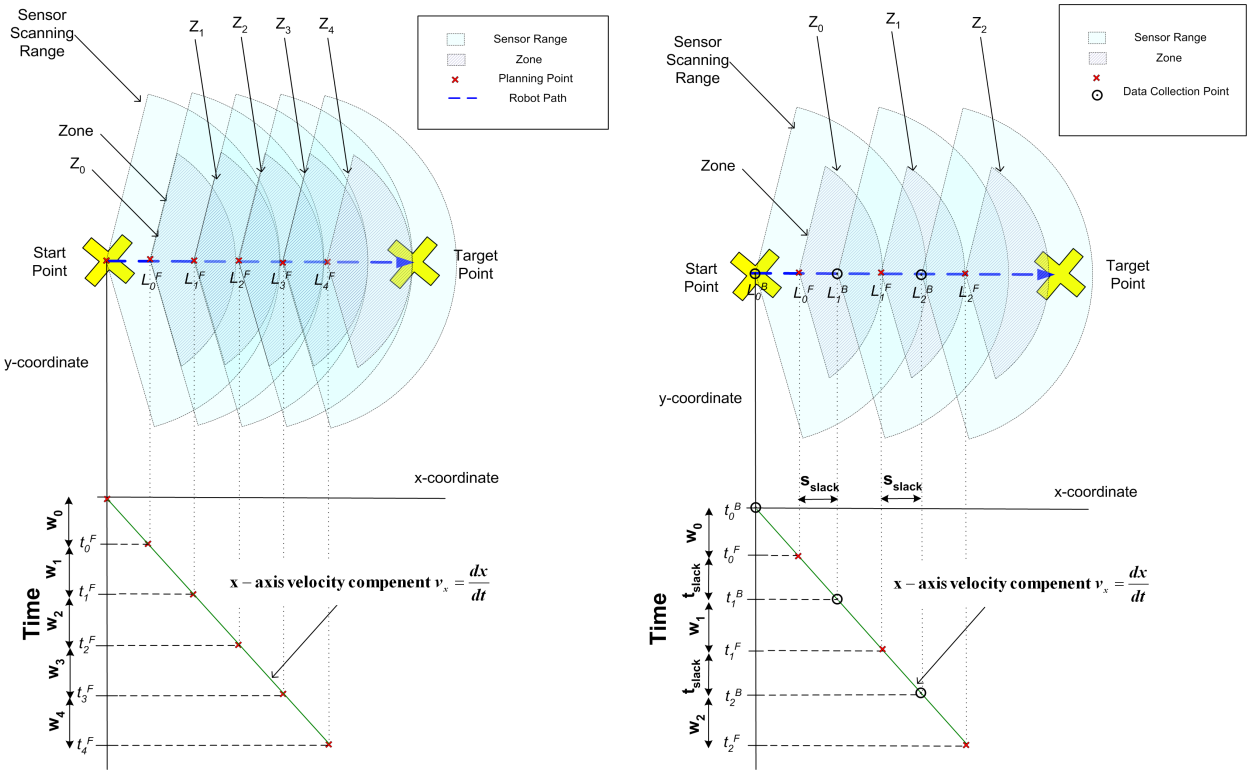
Figure 2(a) demonstrates the division of the platform's path into zones and the division of the associated processing time. If the platform operates at the maximum possible rate then the platform must start collecting sensor information a soon as it finishes planning for the previous zone. As illustrated in Figure 2(a), the platform must start scanning zone Z_{i+1} at point F_i.

In Figure 2(a) the platform starts scanning the next zone as soon as it finishes planning for the current zone. In this case the platform is collecting data, building a surrounding map and planning as fast as possible. While using this approach guarantees that the platform is achieving the best navigational performance, scanning and planning at this fast rate might not be necessary if we can scan at a lower rate and maintain the desired speed. Instead of scanning as fast as possible, we can scan at a rate that is necessary to safely maintain the platform's desired speed. Scanning at a lower rate provides extra time for the processer to execute other



(a) Motionless (b) In motion (c) In motion, safety area included

Figure 1. Zone Abstraction: Zone Boundary



(a) Zone concept: Maximal Scanning (b) Zone concept: Minimal Scanning

Figure 2. Division of robot's path into zones

tasks, which might not have been possible to execute with a maximum scanning rate. At a planning point, the platform has a map describing its intended area of exploration until the boundary of the zone. If we assume a static environment then the platform does not need to start scanning until a point somewhere before the end of the zone. This new point must ensure that the next planning point is at most at the zone boundary.

In this scenario we do not start scanning the next zone at the moment we finished scanning the current zone. Therefore we need to introduce the definition of a *data collection point* to distinguish between the instant the platform starts scanning zone, Z_i, and the instant the platform finishes planning for the previous zone Z_{i-1} because they might not be the same. We define the point (in space and time) at which the platform starts collecting data about its environment from its sensors for zone Z_i as *data collection point* B_i. Using the same methodology used to represent planning points, data collection point B_i can be represented by the tuple (t_i^B, L_i^B), where L_i^B can be represented by the tuple (x_i^B, y_i^B, ψ_i^B). In a two dimensional cartesian coordinate system we can represent B_i by the four parameter tuple $(t_i^B, x_i^B, y_i^B, \psi_i^B)$.

Figure 2(b) demonstrates this case where the platform does not start scanning as soon as the planning for the current zone is done, rather it starts scanning at data collection point B_i for zone Z_i. In this case $W_i = [t_i^B, t_i^F)$, and the processing window duration w can be calculated from Equation (3).

$$w_i = t_i^F - t_i^B \tag{3}$$

We define *zone slack time*, t_{slack}, as the time interval between F_i and B_{i+1}, and the distance moved during this time interval as *zone slack distance*, S_{slack}. If the platform starts data collection for the next zone directly after it finished planing for the current zone then $t_{slack} = 0, S_{slack} = 0$. In this case the platform will be collecting sensor data and planning as fast as possible; Figure 2(a) demonstrates an example of this case. If the platform does not employ maximal scanning then the values for t_{slack} and S_{slack} will not be equal to zero. Equations for calculating the values of t_{slack} and S_{slack} are presented in Section 5.1.2.

The calculation of the desired speed for each zone will be discussed in Section 5.1.2

3.2 Detected Distance Correction

The platform receives each scanning sensor signal while moving toward the target. By the time the platform starts processing the scan signals and planning its move, the platform would have moved further from the points where it collected the signals. This means that the distances recorded at the data collection point are not the same when the platform arrives at the planning point. The difference between the actual distances from its surroundings and the recorded distances from scan signals is dependent on the displacement of the platform since it collected the signals, which in turn

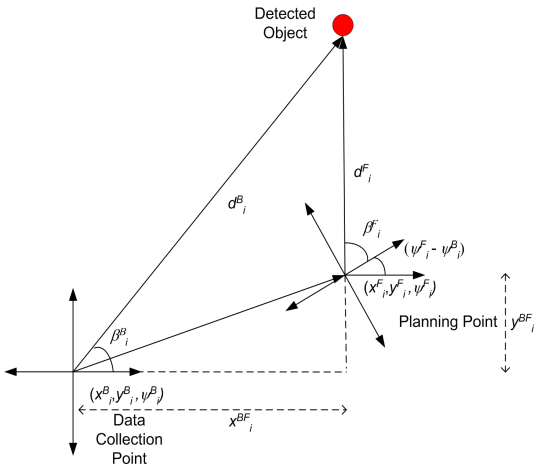


Figure 3. Sensor Signal Correction

is dependent on the platform's velocity and path. This error in the distances and angles can be corrected using standard coordinate transformations. From direct coordinate transformations [5] we can derive Equations (4)—(8) to correct the scan distances

$$d_{ix}^B = d_i^B \cdot cos(\beta_i^B) \tag{4}$$

$$d_{iy}^B = d_i^B \cdot sin(\beta_i^B) \tag{5}$$

$$\begin{bmatrix} d_{ix}^F \\ d_{iy}^F \end{bmatrix} = \begin{bmatrix} \cos(\psi_i^{FB}) & -\sin(\psi_i^{FB}) \\ \sin(\psi_i^{FB}) & \cos(\psi_i^{FB}) \end{bmatrix} \cdot \begin{bmatrix} d_{ix}^B \\ d_{iy}^B \end{bmatrix} + \begin{bmatrix} x_i^{BF} \\ y_i^{BF} \end{bmatrix} \tag{6}$$

$$d_i^F = \sqrt{d_{ix}^F + d_{iy}^F} \tag{7}$$

$$\beta_i^F = \tan 2(d_{ix}^F, d_{iy}^F) \tag{8}$$

where $L_i^B = (x_i^B, y_i^B, \psi_i^B)$ is the point where the sensor signal was received, $L_i^F = (x_i^F, y_i^F, \psi_i^F)$ is the planning point, d_i^B is the distance to the detected object from point L_i^B, d_i^F is the distance to the detected object from the planning point L_i^F, d_{ix}^B, d_{iy}^B are the distance components of d_i^B on the x, y axis respectively, d_{ix}^F, d_{iy}^F are the distance components of d_i^F on the x, y axis respectively, β_i^B, β_i^F are the angles between the vectors d_i^B, d_i^F and the platform axis of orientation respectively, $x_i^{BF} = x_i^B - x_i^F$, $y_i^{BF} = y_i^B - y_i^F$ and $\psi_i^{FB} = \psi_i^F - \psi_i^B$ is the final orientation angle of the platform as shown in Figure 3.

4 Deriving Feasible Processing Windows

For a processing window to be feasible, two conditions must hold. First, the processing window interval must meet the sensor parameter requirements. Second, a sufficient scheduling condition for the scheduling algorithm used must be satisfied. We conjecture that any mobile robotic platform will have a set of tasks $\mathbf{T} = \{\mathbf{T_w} \cup \mathbf{T_{hp}} \cup \mathbf{T_{lp}}\}$, where $\mathbf{T_w}$ is the set of tasks associated with zone processing, $\mathbf{T_{hp}}$ is a (possibly empty) set of periodic tasks with higher priority than $\mathbf{T_w}$ and $\mathbf{T_{lp}}$ is a (possibly empty) set of periodic tasks with lower priority than $\mathbf{T_w}$. In Section 4.1 we present a

general form of the equation used to compute a lower bound for a feasible processing window length. In Section 4.2 we derive bounds on the zone processing window for fixed priority scheduling and combine both bounds.

4.1 Sensor Impact on Processing Window Length

The zone processing window of the platform is dependent on sensor parameters representing delays between sensor readings/invocations, data arrival time, number of sensors, sensor range and sensitivity, and sensor tasks' execution times. Equation (9) is a general equation for deriving the minimum feasible bound on the zone processing window length w. The feasibility function g is a function that is dependent on the sensor(s) and the associated task(s). n is the number of task in $\mathbf{T_w}$, E is the set of execution times for the tasks in $\mathbf{T_w}$ and Δ is the set of delays that might exist between the execution of sensor tasks in $\mathbf{T_w}$.

$$w \geq g(n, E, \Delta) \tag{9}$$

For sensors with adjustable ranges, Δ can be further divided into a set of independent delays, ΔI, that must exist regardless of any other sensor parameters and a set of delays, ΔR, that depend on sensor ranges limitations. Since ΔR is dependent on the sensor ranges, we can insert the set of effective ranges of platform sensors, R, directly as a parameter in the function g. Unfortunately the function g is application dependant and must be derived separately for each application. In Section 7 we derive the function g for the platform we chose for evaluation.

$$w \geq g(n, E, \Delta I, R) \tag{10}$$

4.2 Fixed Priority Bound Derivation

In this section we derive a lower bound on the periods for the tasks running on the processor based on fixed priority scheduling and time demand analysis [1]. In this work we assume that the tasks in $\mathbf{T_{hp}}$ are independent and preemptive with deadlines equal to periods. Throughout the remainder of this paper we will assume, without loss of generality, that the tasks in task set $\mathbf{T}$ are ordered according to their priority such that if $i < j$ then i has a higher priority than j.

For any task in $\mathbf{T_{hp}}$, the lower bound on the period can be calculated from Equation (11), where p_j is the period and e_j is the worst case execution time of task T_j.

$$p_j \geq e_j + \sum_{i=1}^{j-1} \left\lceil \frac{p_j}{p_i} \right\rceil \cdot e_i \tag{11}$$

Because $\frac{p_j}{p_i} \leq \left\lceil \frac{p_j}{p_i} \right\rceil \leq \frac{p_j}{p_i} + 1$ we can substitute $\frac{p_j}{p_i} + 1$ for $\left\lceil \frac{p_j}{p_i} \right\rceil$ to get a looser, more pessimistic bound, such that

$$\begin{aligned}
p_j &\geq e_j + \sum_{i=1}^{j-1} \left(\frac{p_j}{p_i} + 1 \right) \cdot e_i \\
p_j &\geq e_j + p_j \sum_{i=1}^{j-1} \frac{e_i}{p_i} + \sum_{i=1}^{j-1} e_i \\
p_j \left(1 - \sum_{i=1}^{j-1} \frac{e_i}{p_i} \right) &\geq \sum_{i=1}^{j} e_i \\
p_j &\geq \frac{\sum_{i=1}^{j} e_i}{1 - \sum_{i=1}^{j-1} \frac{e_i}{p_i}}
\end{aligned} \tag{12}$$

Using Equation (12) we can calculate a lower bound for each task period in $\mathbf{T_{hp}}$ that results in a schedulable task set. For the task set $\mathbf{T_w}$, however, Equation (11) cannot be directly extended to derive a lower bound for the processing window w. Instead we must combine Equation (10) and Equation (11) to account for compulsory delays between tasks in $\mathbf{T_w}$ because certain types of sensors, such as sonar or ultrasonic sensors, must have a minimum delay between sending any two signals due to signal interference or crosstalk. Thus, Equation (13) combines both bounds.

$$w \geq g(n, E, \Delta I, R) + \sum_{T_j \in \mathbf{T_{hp}}} \left\lceil \frac{w}{p_j} \right\rceil \cdot e_j \tag{13}$$

Equation (13) is a conservative overestimate of the lower bound on w because not every task in $\mathbf{T_{hp}}$ will interfere with each task in $\mathbf{T_w}$ every time it is released. Using the same substitution we used in Equation (11) we get Equation (14).

$$w \geq \frac{g(n, E, \Delta I, R) + \sum_{T_j \in \mathbf{T_{hp}}} e_j}{1 - \sum_{T_j \in \mathbf{T_{hp}}} \frac{e_j}{p_j}} \tag{14}$$

If $\mathbf{T_{lp}} \neq \varnothing$ then we calculate the processing window by assigning the tasks in $\mathbf{T_{lp}}$ periods that are integer multiples of w. Thus, assuming $\mathbf{T_{lp}}$ contains m tasks, we will calculate a lower bound on the period for the tasks in $\mathbf{T_{lp}}$, $p_{lp} = \alpha \cdot w$ where α is an integer number.

Using Equation (11) for p_{lp} we get

$$\alpha \cdot w \geq \sum_{T_j \in \mathbf{T_{lp}}} e_j + \left\lceil \frac{\alpha \cdot w}{w} \right\rceil \cdot g(n, E, \Delta I, R) + \sum_{T_j \in \mathbf{T_{hp}}} \left\lceil \frac{\alpha \cdot w}{p_j} \right\rceil \cdot e_j \tag{15}$$

Because $\frac{\alpha \cdot w}{p_j} \leq \left\lceil \frac{\alpha \cdot w}{p_j} \right\rceil \leq \frac{\alpha \cdot w}{p_j} + 1$ we can substitute $\frac{\alpha \cdot w}{p_j} + 1$ for $\left\lceil \frac{\alpha \cdot w}{p_j} \right\rceil$ to get a looser, more pessimistic bound, such that

$$\alpha \cdot w \geq \sum_{T_j \in \mathbf{T_{lp}}} e_j + \lceil \alpha \rceil \cdot g(n, E, \Delta I, R) + \sum_{T_j \in \mathbf{T_{hp}}} \left(\frac{\alpha \cdot w}{p_j} + 1 \right) \cdot e_j \tag{16}$$

$$\alpha \geq \frac{1}{w} \sum_{T_j \in \mathbf{T_{lp}}} e_j + \frac{\lceil \alpha \rceil \cdot g(n, E, \Delta I, R)}{w}$$

$$+\frac{1}{w}\sum_{T_j\in\mathbf{T_{hp}}}\left(\frac{\alpha\cdot w}{p_j}+1\right)\cdot e_j \tag{17}$$

Because α is an integer can substitute $\lceil\alpha\rceil$ for α. Thus

$$\alpha \geq \frac{\frac{1}{w}\sum_{T_j\in\mathbf{T_{lp}},\mathbf{T_{hp}}} e_j}{1-\frac{g(n,E,\Delta I,R)}{w}+\sum_{T_j\in\mathbf{T_{hp}}}\frac{e_j}{p_j}} \tag{18}$$

Substituting for w from Equation (14), we get Equation (19).

$$\alpha \geq \frac{\sum_{T_j\in\mathbf{T_{lp}},\mathbf{T_{hp}}} e_j}{\sum_{T_j\in\mathbf{T_{hp}}} e_j} \tag{19}$$

But since we limited α to integer values we must take the ceiling of the right side of Equation (19). Therefore

$$\alpha \geq \left\lceil\frac{\sum_{T_j\in\mathbf{T_{lp}},\mathbf{T_{hp}}} e_j}{\sum_{T_j\in\mathbf{T_{hp}}} e_j}\right\rceil \tag{20}$$

This method minimizes w while calculating a bound on the period for the tasks in $\mathbf{T_{lp}}$. However, it is possible to achieve a lower period of the tasks in $\mathbf{T_{lp}}$ if we assign a higher value for w. To do that, we fix the value of α and solve Equation (17) for w and take the minimum value of w that satisfies both Equation (16) and Equation (14).

In this section we computed sufficient lower bounds for periods and processing windows such that the tasks in $\mathbf{T_w}$, $\mathbf{T_{hp}}$ and $\mathbf{T_{lp}}$ can be guaranteed to meet their deadlines.

5 Environmental Impact on Speed and Sampling Rates

The platform depends on sensors to plan its path and to determine the presence of obstacles and their distance. The maximum speed at which the platform can travel is related to the rate the environment signals can be scanned and processed. If the platform moves faster than the sensor signals can be processed, then the motion will be unsafe because there might be an obstacle in the path that will be undetected at that rate. In Section 5.1.2 we derive upper bounds on the platform's speed throughout a zone with ideal assumptions with regards to speed transition time, in Section 5.2 we relax the ideal assumptions and derive approximation formulas for the speed transition time and in Section 5.3 we present an algorithm for adjusting the zone processing window to increase platform speed.

5.1 Zone Speed Choice

The speed of the platform for a zone is dependent on the radius of the zone, the zone-processing window, the speed of the platform in the previous zone and the existence of obstacles in the zone. We derive the calculation of the upper bound on the desired speed for the zone, $v_{max\,i}$, in two distinct cases: an obstacle free environment and an environment in which obstacles exist.

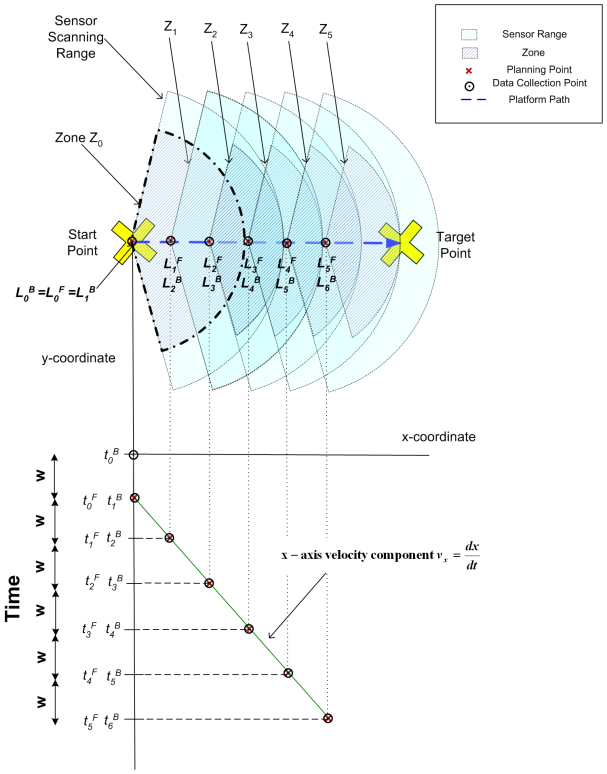


Figure 4. Maximal scanning with no obstacles

5.1.1 Obstacle Free Environment

We first calculate the upper bound on the zone speed for the maximal sensor scanning scenario (i.e., scanning as fast as possible). Figure 4 demonstrates this scenario. Initially the platform starts scanning its intended area of exploration at point $B_0 = (0, x_0^B, y_0^B, \psi_0^B)$. Because it takes the platform w time units to finish collecting the sonar data and planning the path, it is not safe for the platform to start moving until $t = w$. Therefore the first planning point F_0 will have the same position coordinates as the first data collection point B_0: $F_0 = (w, x_0^B, y_0^B, \psi_0^B)$. Beyond the first zone, planning points for the current zone and data collection points for the next zone will have the same time and position, $B_{i+1} = F_i$.

In each zone the platform can travel a maximum distance equal to the zone radius D_i before entering another zone. The platform also must spend at least w time units in the zone because that is the time interval the platform takes for collecting sensor data and planning the path for zone Z_i. Therefore if we assume constant speed through zone Z_i, the maximum speed the platform can travel safely through zone Z_i while being able to collect sensor data and plan for zone Z_{i+1} can be calculated from Equation (21).

$$v_{max} = \frac{D_i}{w_{i+1}} \tag{21}$$

The zone radius D_i can be calculated from Equation (1). If we assume the platform is traveling at the maximum possible speed v_{max} then the distance the platform moves be-

tween points B_i and F_i is equal to $v_{max} \cdot w_i$. Therefore the zone radius can be calculated from Equation (22).

$$D_i = r - v_{max} \cdot w_i - S_M \tag{22}$$

Substituting Equation (22) in Equation (21) and solving for v_{max} we get

$$v_{max} = \frac{r - v_{max} \cdot w_i - S_M}{w_{i+1}}. \tag{23}$$

if we assume a constant processing window $w_i = w_{i+1} = w$ then Equation (23) becomes Equation (25)

$$v_{max} = \frac{r - v_{max} \cdot w - S_M}{w} \tag{24}$$

$$= \frac{r - S_M}{w} - v_{max} = \frac{r - S_M}{2 \cdot w}. \tag{25}$$

If at any plan point F_i we change the zone processing window w_i or change the sensor detection range r_i, then Equation (23) becomes

$$v_{maxi} = \frac{r_i - v_{i-1} \cdot w_i - S_M}{w_{i+1}}, \tag{26}$$

where v_{maxi} is the speed for zone Z_i, w_{i+1} is the processing window for Z_{i+1}, r_i, v_i and w_i are the sensor detection distance, speed and processing window for Z_i respectively.

If the platform does not employ maximal scanning then the values of S_{slack} and t_{slack} can be calculated from Equation (27) and Equation (28) respectively. It is clear from Figure 2(b) that the zone slack distance is less than the zone radius by a distance of $v_i \cdot w_{i+1}$ because the platform must scan for the next zone Z_{i+1} before it enters the zone (i.e., while the platform is still traveling in zone Z_i) in order to achieve continuous motion.

$$S_{slack} = r - (v_i \cdot w_{i+1} + v_{i-1} \cdot w_i + S_M) \tag{27}$$

$$t_{slack} = \frac{S_{slack}}{v_i} \tag{28}$$

5.1.2 Obstacles Exist

If an obstacle exits then the distance the platform can safely move is not the zone radius, but rather the distance between the obstacle and the platform, X_{obs}. Therefore if $X_{obs} < D_i$ the platform speed can be calculated from

$$v_{maxi} = \frac{X_{obs}}{w_{i+1}}. \tag{29}$$

If the platform is not using maximal scanning then the platform might switch to maximal scanning if it encounters an obstacle because it needs to maintain a higher speed or scan in a different direction in order to explore alternative paths, as shown in Figure 5.

Figure 5 demonstrates the relation between zones, sensing range, zone processing window and the extra distance the platform moves without scanning the surroundings

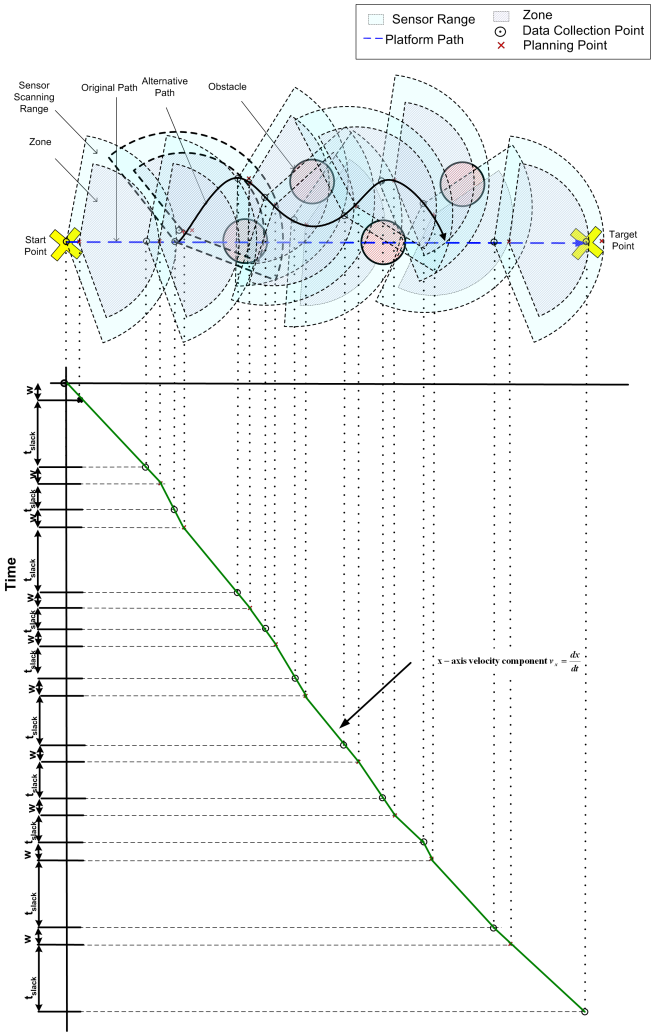


Figure 5. Exitance of obstacles in the environment

(zone slack distance S_{slack}) in the exitance of obstacles. In this scenario the platform's original path to its target point is a straight line. The platform starts scanning the path area using an initial value for its sensing range and processing window. The platform scans as slowly as possible to be able to use spare processing capacity for other tasks running on the processor. But as the platform faces its first obstacle in the path, the platform needs to scan at a faster rate because more scans are needed at shorter obstacle distances to determine the alternative path. As the platform starts scanning at a faster rate, the zones overlap more and the zone slack distance that the platform moves without scanning becomes smaller or non-existent. In this scenario the platform faced more obstacles in its alternative path. Therefore it needed to keep scanning at a higher rate until it reached a path clear from obstacles after the fourth obstacle.

5.2 Speed Transition Time

In the previous section we considered the ideal case in which the robot can switch between two speeds instantaneously. In a real system the speed transition time interval

is not zero and the speed function takes the form of a decaying or a rising exponential function instead of a step function. Therefore if we have an obstacle, the final maximum speed would not be given by Equation (29), rather the final maximum speed will be given by Equation (30), where

$f(v_{maxi}, v_{i-1}, w_i)$ is a function that describes the speed during the transition time interval.

$$X_{obs} = f(v_{maxi}, v_{i-1}, w_{i+1}) \tag{30}$$

v_{maxi} can be calculated by solving Equation (30) for v_{maxi}. If we assume that the system traverses the environment at a constant speed unless it switches between two speeds then we can derive the formula for the function f.

The initial platform speed V_1, distance to the obstacle X_{obs} and processing window length w are known, while both final speed V_2 and the instant the speed reached V_2 are both unknown. Therefore the distance to the obstacle X_{obs} is given by Equation (31), where τ is the instant the platform reduces its speed and $v(t)$ be the platform velocity as a function of time. Because the platform will only change its speed at a planning point, τ will be at the beginning of a processing window.

$$X_{obs} = \int_{\tau}^{\tau+w} v(t)dt. \tag{31}$$

Let t_s denote the instant the speed reaches its final value V_2. Let $v_s(t)$ be the speed during the transition time interval $[\tau, \tau+t_s)$ given as a function of time. We get Equation (32).

$$X_{obs} = \int_{\tau}^{\tau+t_s} v_s(t)dt + \int_{\tau+t_s}^{\tau+w} V_2 dt \tag{32}$$

Since the initial and final velocities are variables for each processing window, they can be parameters of $v_s(t)$. Therefore the speed during the transition interval $[\tau, \tau + t_s)$ can be given as $v_s(t, V_1, V_2)$, where V_1 is the initial speed and V_2 is the final speed. Because the existence of an obstacle in the environment will imply reducing speed $V_2 = v_{maxi}$ and the initial velocity will be equal to the platform velocity in the pervious zone, $V_2 = v_{maxi}$. Therefore Equation (32) becomes

$$X_{obs} = \int_{\tau}^{\tau+t_s} v_s(t, v_{i-1}, v_{maxi})dt + \int_{\tau+t_s}^{\tau+w} v_{maxi} dt \tag{33}$$

From Equation (33) we deduce the formula for the function f

$$f(v_{maxi}, v_{i-1}, w_{i+1}) = \int_{\tau}^{\tau+t_s} v_s(t, v_{i-1}, v_{maxi})dt + \int_{\tau+t_s}^{\tau+w_{i+1}} v_{maxi} dt \tag{34}$$

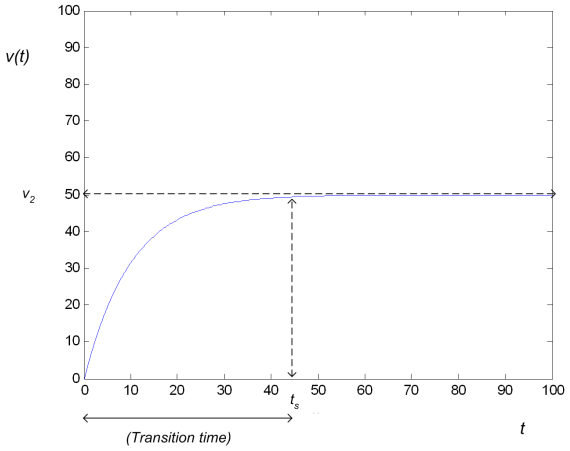


Figure 6. Ideal response

In Section 5.2.1 we derive an exponential approximation formula for the function f. In Section 5.2 we derive an linear approximation formula for the function f, and in Section 5.2.3 we compare both models.

5.2.1 Exponential Approximation Model

A good controller design aims at eliminating oscillatory components and overshoot in the response signal and achieving a smooth rising exponential response until steady state is reached (or a smooth decaying exponential from a steady state) as shown in Figure 6. We will use this response to calculate speed transition time. Figure 6 describes a function with the conditions $v(0) = 0, V(\infty) = V_2$, since we know that the function is exponential, the response function can be modelled by Equation (35).

$$v(t) = V_2 \cdot (1 - e^{-Q \cdot t}), Q > 0 \tag{35}$$

Where Q is a parameter determined by fitting the speed controller response data to Equation (35).

If the initial speed was not zero, but V_1, and the speed transition occurred at time instant τ, Equation (35) becomes Equation (36)

$$v(t) = V_1 + V_2 \cdot (1 - e^{-Q \cdot (t-\tau)}), Q > 0 \tag{36}$$

In the case of reducing speed, the response function has the conditions $v(-\infty) = V1, V(\infty) = 0$ and we want $v(0) \simeq V1$. Equation (37) models the response to these conditions.

$$v(t) = V_1 \cdot (1 - A \cdot e^{F \cdot t}), F > 0, 0 < A < 1 \tag{37}$$

If the speed transition occurs at time τ, then Equation (37) becomes

$$v(t) = V_1 \cdot (1 - A \cdot e^{F \cdot (t-\tau)}), F > 0, 0 < A < 1 \tag{38}$$

Parameters A, F are determined by fitting the speed controller response data to Equation (38).

Substituting for $v_s(t)$ in Equation (32) from Equation (38) we get Equation (39).

$$X_{obs} = \int_{\tau}^{\tau+t_s} V_1 \cdot (1 - Ae^{F(t-\tau)})dt + \int_{\tau+t_s}^{\tau+w} V_2 dt$$
$$= V_1 \cdot \left(t_s + \frac{A \cdot (1 - e^{F \cdot t_s})}{F}\right) + V_2 \cdot (w - t_s) \quad (39)$$

At the time instant $t_s + \tau$ the speed reaches V_2. Substituting $t_s + \tau$ in Equation (38) we get

$$V_2 = V_1 \cdot (1 - Ae^{F \cdot (t_s)}) \Rightarrow t_s = \frac{1}{F} \cdot \ln\left(\frac{1}{A}\left(1 - \frac{V_2}{V_1}\right)\right) \quad (40)$$

Substituting t_s from Equation (40) in Equation (39) we get Equation (41).

$$X_{obs} = \frac{1}{F} \cdot \left((V_1 - V_2) \cdot \ln\left(\frac{1}{A}\left(1 - \frac{V_2}{V_1}\right)\right) + V_2 \cdot (1 + wF) + V_1 \cdot (1 + A)\right) \quad (41)$$

Equation (41) can be solved for V_2 using a numerical iterative methods only, which are too computationally expensive for a real-time system. Therefore in the next section we drive a linear approximation model with a deterministic solution.

5.2.2 Linear Approximation Model

A linear approximation of the speed function during the transition time generates a deterministic equation that can be solved to find the value of the desired speed. Figure 7 shows the linear approximation for the speed which is computed using Equation (42).

$$v(t, V_1, V_2) = \begin{cases} \left(\frac{V_2 - V_1}{t_s}\right) \cdot t & -\tau \cdot \left(\frac{V_2 - V_1}{t_s}\right) + V_1 \\ & V_1 \neq V_2,\ V_1, V_2 > 0 \\ V_1 & V_2 = V_1,\ V_1, V_2 > 0 \end{cases} \quad (42)$$

Substituting Equation (42) in Equation (31) we get Equation (43).

$$X_{obs} = \int_{\tau}^{\tau+t_s} \left(\frac{V_2 - V_1}{t_s}\right) \cdot t \ - \tau \cdot \left(\frac{V_2 - V_1}{t_s}\right) + V_1 \, dt + \int_{\tau+t_s}^{\tau+w} V_2 \, dt. \quad (43)$$

The transition instant τ adds only a shift to the equation and does not change the outcome. Therefore we can substitute $\tau = 0$ in Equation (43) to get Equation (44)

$$X_{obs} = \int_{0}^{t_s} \left(\frac{V_2 - V_1}{t_s}\right) \cdot t + V_1 \, dt + \int_{t_s}^{w} V_2 \, dt$$

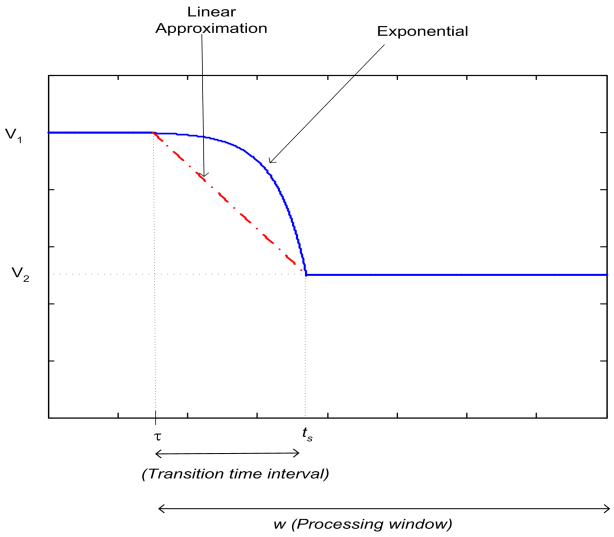


Figure 7. Exponential and linear approximation of transition time

$$= \frac{(V_2 - V_1)t_s}{2} + V_1 t_s + V_2(w - t_s) \quad (44)$$

If we fit the platform controller response data into a linear equation of the form $x = mt + b$ we can find the slope value m. From Equation (42), $m = (V_2 - V_1)/t_s \Longrightarrow t_s = (V_2 - V_1)/m$. Substituting the value of t_s in Equation (44) we get

$$X_{obs} = \frac{-V_2^2}{2m} + V_2\left(w + \frac{V_1}{m}\right) - \frac{V_1^2}{2m} \quad (45)$$

Solving Equation (45) for V_2 we get Equation (46)

$$V_2 = V_1 + wm \pm \sqrt{2V_1 wm + w^2 m^2 - 2X_{obs} m} \quad (46)$$

Only one solution of Equation (46) will be in our desired range of $0 \leq V_2 \leq V_1$.

5.2.3 Linear vs Exponential

Figure 8 shows a comparison between the desired speed value with an initial speed of 50 cm/s using the exponential model, linear model, and zero transition time (ideal) to model the speed transition function. The solutions for the exponential model equation were obtained iteratively for each point on the graph using Newton's method in Matlab.

The values in Figure 8 are calculated for a corrected obstacle distance range of one meter (The corrected obstacle distance = obstacle distance -safe stopping distance.) Actual speed data from the same platform that we used for our experimental evaluations[1] was fit to Equation (38) and Equation (42) using the method of least squares in Matlab.

We can see from Figure 8 that the linear model provides a good approximation for transition time. We have used the

[1]Please refer to Section 7 for more details about the platform used in the experiments.

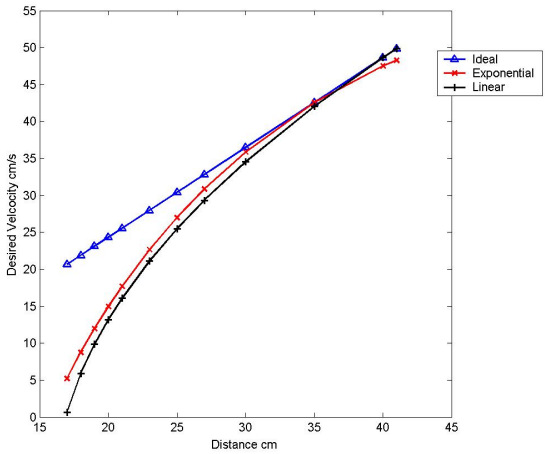


Figure 8. Transition time model comparison

linear approximation model in our experiments because of its deterministic calculation time. While it is possible to obtain offline solutions for the exponential model for a set of initial speeds and a range of obstacle distances with a fixed processing window, it would be difficult if the processing window length was variable because we have to generate a table for each range of w. Considering that the difference between the exponential and linear models is within an acceptable range, it will be more feasible to use the linear model instead of a lookup table for the exponential model.

5.3 Processing Window Adjustment

The speed of the platform and the duration of the processing window are interdependent. In the previous section, we assumed a fixed processing window and computed limits on the speed at which the platform can move through the environment. In this section, we show that an alternative is to adjust the processing window in an attempt to travel as fast as possible around an obstacle.

The processing window is adjusted according to the algorithm shown in Figure 9, but only at planning points. The algorithm starts by setting the sensor detection range r to the maximum sensor detection range. The upper bound on the platform speed v_{max} is set as if no obstacles exist in the platform's path (as explained in Section 5.1.2). The initial value is set to its desired speed as long as that value is less than or equal to v_{max}.

At the end of the zone processing window, there will be two cases: either there is an obstacle in the path or the path is obstacle free.

Case 1: An obstacle exists. Even though the existence of an obstacle will probably cause the value of v_{max} to drop, the desired speed might still be less than or equal to v_{max}. If so, set the speed of the platform to the desired speed.

However, if the desired speed is greater than v_{max} and there is a possibility to increase the speed by reducing the sensor detection distance (which in turn reduces sensor delays and therefore w), then a new value is calculated for w. Next a new speed is calculated for the platform, and the zone slack time, t_{slack}, is set to zero.

Task	e (ms)	p (ms)	Priority
Dead Reckoning	5	17	1
PID	1	50	2

Table 1. Task parameters

Case 2: No obstacle exists. If no obstacle exists and the current speed is equal to the desired speed, there is no need to make a change. The algorithm simply follows the process described in the previous sections to compute v_{max} and w.

If the current speed is less than the desired speed, we have to try to maximize the platform's speed by choosing the optimal value for the detection range r that would result in the maximum increase in speed. However, if the calculated value for r is not within a valid range, we use the lowest sensor range r_{min} that will give the maximum possible speed (less than or equal to the desired speed). Each time r is adjusted, zone slack time t_{slack} must be calculated based on the new values for r and w.

6 Simulation

Before implementing the algorithm on a real mobile robotic platform we have constructed a simulation to evaluate the performance speed adjustment algorithm using Matlab. The simulation assumes the linear speed transition model presented in 5.2.2, earlier simulation results that assumes ideal speed transition conditions (i.e., no speed transition time when switching between speeds) have been published in [20]. The simulation uses the parameters of an autonomous mobile robot that was used as the lead robot in the Robotic Safety Marker project [8, 21, 19]. The robot has 24 sonar sensors arranged in a ring. Using these sensors the robot builds a map of its environment and determines the distance to any obstacles in its path.

Each sonar sensor is associated with two tasks with different delays between their execution: a sonar send task that sends the sonar signal from the sensor and a sonar receive task that checks the sensor for the received signal and calculates the distance to the objects in the sensor direction. In addition, two more tasks are associated with the zone processing window: 1) a map task generates a map for the platform's surroundings based on the sensor data, 2) a plan task that processes the generated map and plans the platforms path and speed for the next zone. Deriving lower bounds on the periods for these tasks was discussed in [19], but without the processing window abstraction. Moreover, due to differences in hardware between the current platform and the robot used in [19], the feasibility function g for the sonar sensors is slightly different from the feasibility function presented in [19]. For the current sonar senor set, the feasibility

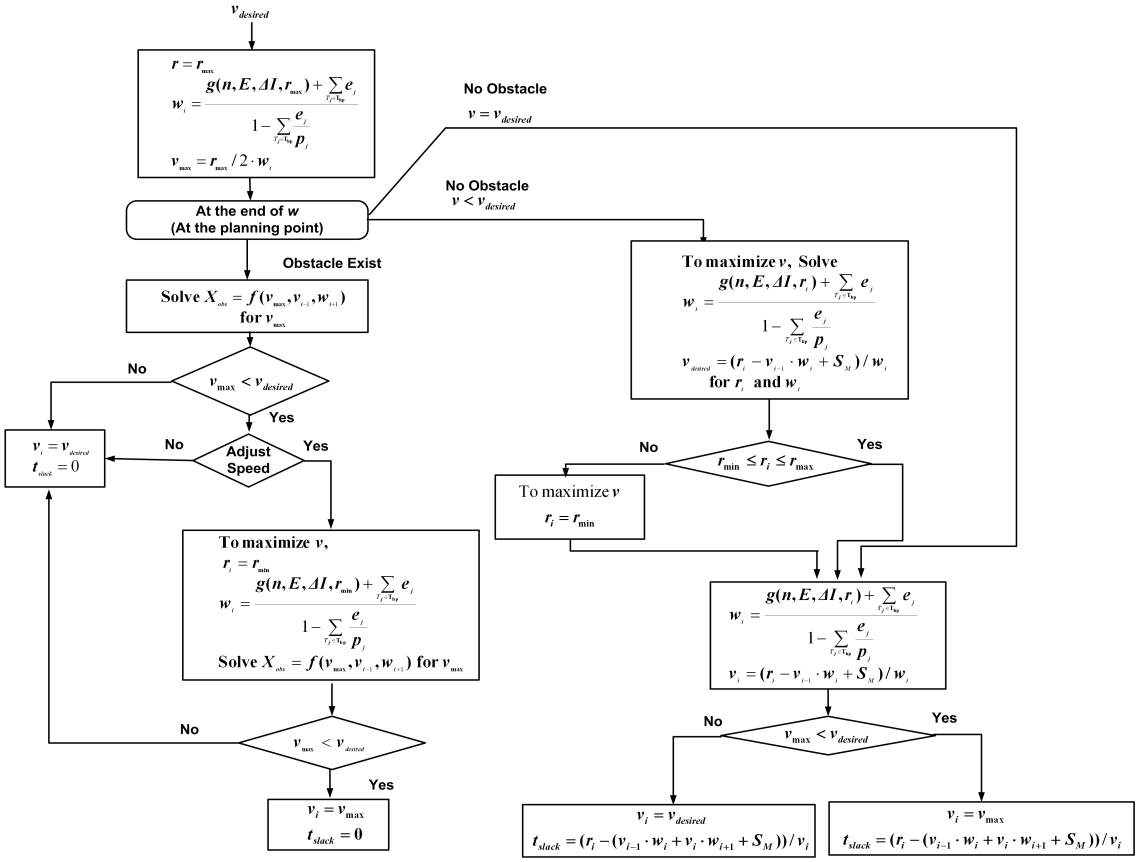


Figure 9. Speed-processing window adjustment algorithm

Ideal Speed Transition	Without processing window adjustment	With processing window adjustment
t_{total} (s)	96.48	73.17
$\bar{v}$ (cm/s)	38.20	48.02
$\bar{v}/v_{desired}$ (%)	76.40%	96.04%
$t_{slack_{total}}$ (s)	50.14	66.28
$\overline{t_{slack}}$ (s)	4.69	6.83
$t_{slack_{total}}/t_{total}$(%)	51.97%	90.57%
Linear Speed Transition	Without processing window adjustment	With processing window adjustment
t_{total} (s)	102.26	75.31
$\bar{v}$ (cm/s)	35.16	43.52
$\bar{v}/v_{desired}$ (%)	70.30%	87.04%
$t_{slack_{total}}$ (s)	47.55	64.29
$\overline{t_{slack}}$ (s)	4.11	6.28
$t_{slack_{total}}/t_{total}$(%)	46.50%	85.52%

Table 2. Simulation results summary

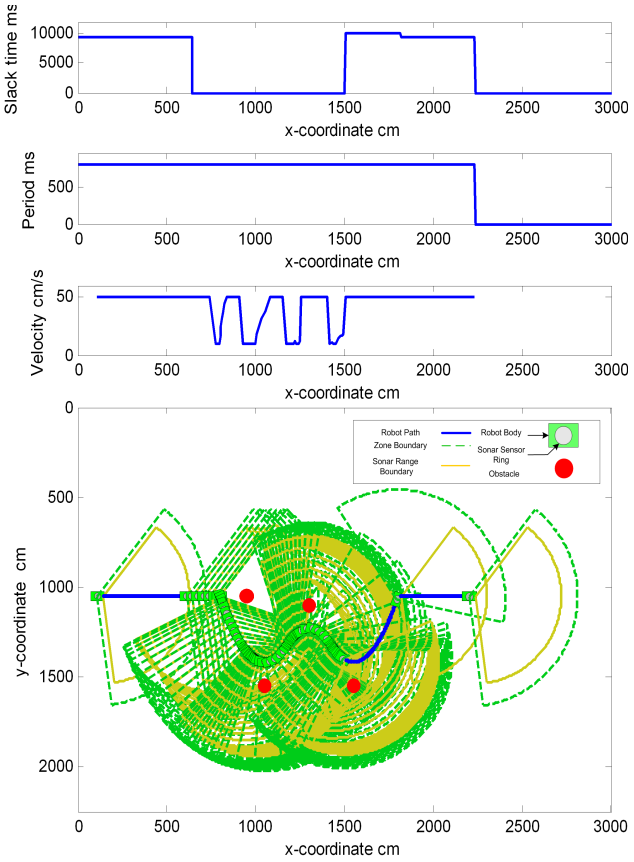


Figure 10. Test with no processing window adjustments

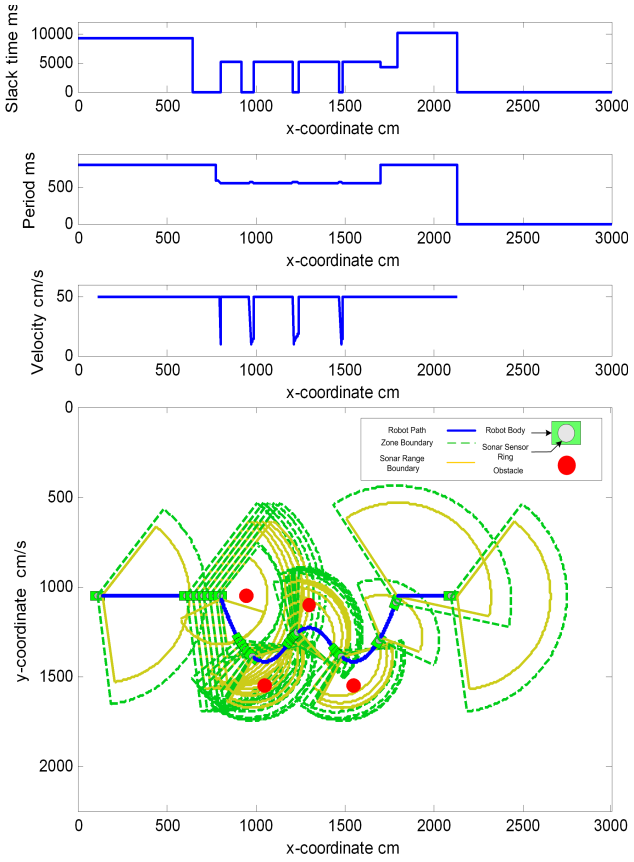


Figure 11. Test with processing window adjustments

function g is given by Equation (47)

$$g(n, E, \Delta I, R) = n \cdot (e_{send} + e_{recv} + \tau + \frac{2 \cdot r}{340}) + e_{map} + e_{plan} \quad (47)$$

n is the number of sonar sensors used; τ is the delay used to eliminate crosstalk between a received sonar signal and the next sonar send signal; e_{send} is the execution time of a sonar send task; e_{recv} is the execution time of a sonar receive task; e_{map} is the the map execution time of the map task, e_{plan} is the execution time of the plan task and r is the sonar sensor range. The set of higher priority tasks, $\mathbf{T_{hp}}$, are given in Table 1. These tasks are a *PID* task that controls the robot motors and a *Dead Reckoning* task that calculates the robot's coordinates based on dead reckoning techniques.

The simulation environment is event based and simulates a 30m x 22.5m space where the robot moves. The space matrix is projected onto a visualization image where each pixel represents 1 cm x 1 cm of space.

Because one of the goals of this research is the automatic adjustment of speed and processing windows for each zone, no obstacle avoidance algorithm was used. Instead the robot follows a path that maintains a safe distance of 60 cm from obstacles. This scenario demonstrates how the existence of obstacles in the platform's path affects the zone processing window and speed.

The desired speed for the robot in this simulation is 50 cm/s. The robot is using 10 sonar sensors out of its 24 sensors to build its environment map (a smaller number of sonar sensors is used in order to reduce crosstalk effects).

Figures 10 and 11 show the simulation of the robot moving along the path showing both zones and actual sonar range on the robot's path. Figure 10 shows the simulation of the robot traversing the path without using the speed adjustment, while Figure 11 shows the simulation of the robot traversing the path using the processing window adjustment. The figures show location of the robot at each data collection point B_i while the zones start and finish at the planning points F_i. The figures also show robot velocity and processing window plotted against the x-coordinate of the path.

We can see in Figure 11 that the robot adjusts its sonar range as it gets closer to the obstacle in order to adjust its processing window and maximize its speed. The simulation also shows that the robot switches its scanning rate to maximal scanning as it faces an obstacle.

Table 2 shows a comparison between both cases in terms of total time t_{total} needed to traverse the path, average speed $\bar{v}$, the ratio of average speed to the desired speed $\bar{v}/v_{desired}$, total slack time $t_{slacktotal}$ over the whole path, $\overline{t_{slack}}$ average slack time over the whole path, and $t_{slacktotal}/t_{total}$ is the ratio of the total slack time to the total time needed to complete the path. The result shows that the processing window adjustment algorithm improved the average speed for

the robot over the whole path by 16.74% relative to the desired speed assuming a linear for approximating speed transitions and 19.96% assuming ideal speed transitions [20]. The simulation result also shows that there is more slack time gained by using the processing window adjustment algorithm.

7 Experimental Results

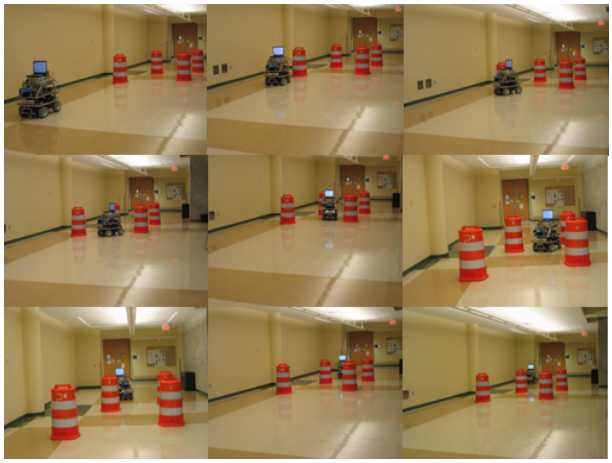


Figure 12. Test Progress Pictures

We also evaluated the processing window adjustment algorithm on the actual robot described in Section 6. The test scenario is similar to the simulated scenario, but we have adopted a linear approximation of the speed transition function of Equation (30), described in detail in Section 5.2.2, to account for the speed switching delay due to robot hardware.

Because one of the goals of this research is the automatic adjustment of speed and processing windows for each zone, the platform was steered manually through its path, moving closer to objects than the path planning algorithm would.

Figure 12 shows a series of pictures taken during the actual test demonstrating the progress of the robot in its path to the target. Figure 13 shows the actual path the platform took to its target point with speed adjustments, but no processing window adjustment. Figure 15 shows the platform's path to its target, which is approximately the same as the path in Figure 13, but this time we allowed both the speed and the processing window to be adjusted, using the algorithm in Figure 9. Figures 13 and 15 also show the zones on the robot's path. Figures 14 and 16 are the same as Figures 13 and 15 respectively but they show the both zones and actual sonar range on the robot's path (Recall from Section 3.1 that the zones are smaller than the sensing range due to added safety area and motion during sensing). We can see from the figures that the platform reduces its speed as it encounters an obstacle in order to meet its processing deadlines. Figure 15 demonstrates the improvement gained from the processing window adjustment algorithm in terms of higher platform speed in the obstacle region; the speed in

Figure 13. Zones plotted on robot path with no processing

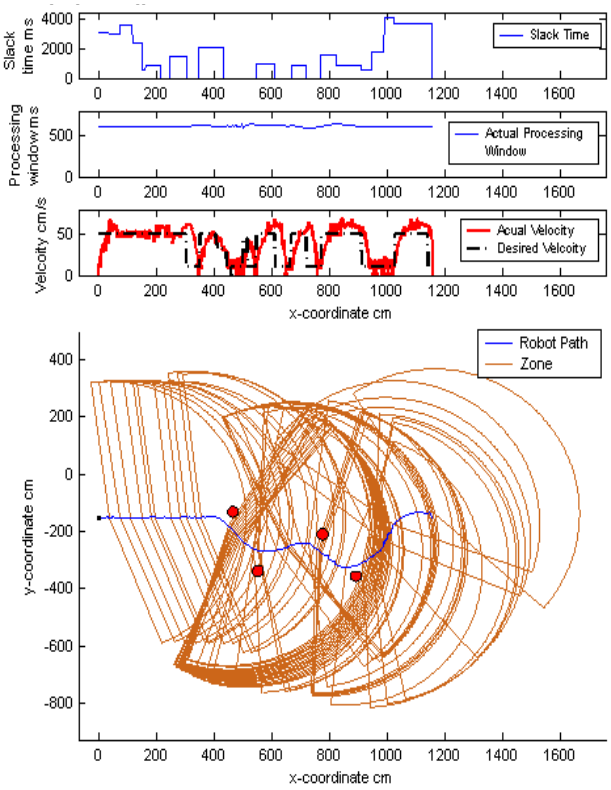
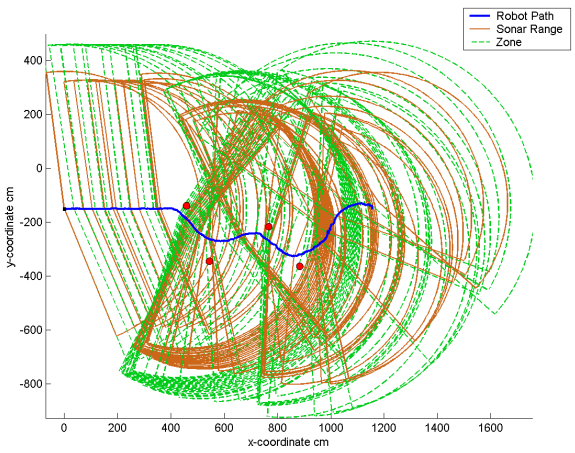


Figure 14. Zone and sonar range plotted on robot path



these intervals when the platform detects a nearby obstacle is higher than their counterpart in Figure 13. [2]

We can see in Figure 13 that the zones are all the same size because there is no processing window adjustment. We note that when the robot gets closer to the obstacles, the robot scans at faster a rate. Thus the zone slack distance and time become either shorter or equal to zero. In Figure 15

[2]Note that the coordinates where the platform detects the obstacle and reduces its speed are not exactly the same because the platform paths are approximately the same but not exactly the same.

	Without processing window adjustment	With processing window adjustment
t_{total} (s)	85.20	63.53
$\bar{v}$ (cm/s)	29.74	36.09
$\overline{v_{actual}}$ (cm/s)	29.97	36.85
$\bar{v}/v_{desired}$ (%)	59.48%	72.18%
$\overline{v_{actual}}/v_{desired}$(%)	59.97%	73.7%
$t_{slacktotal}$ (s)	25.07	32.72
$\overline{t_{slack}}$ (s)	1.65	.76
$t_{slacktotal}/t_{total}$(%)	39.46%	51.79%

Table 3. Experimental results summary

Figure 15. Zones plotted on robot path with processing

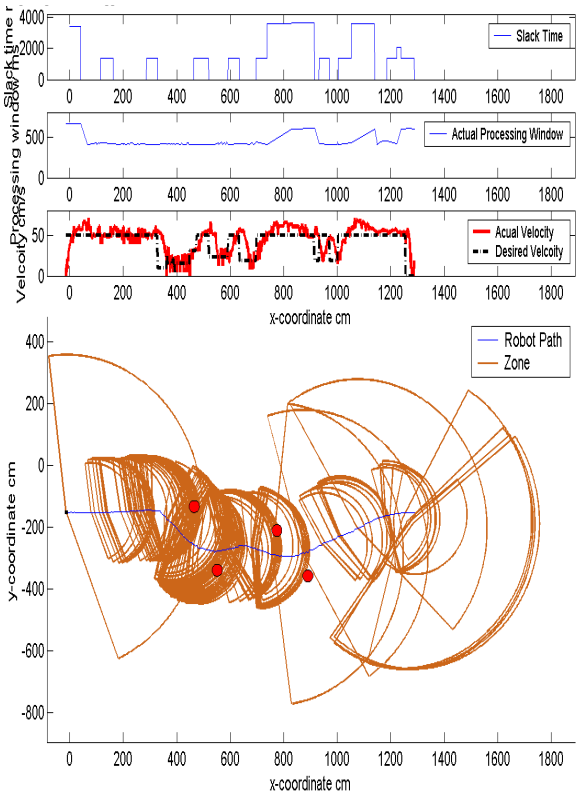
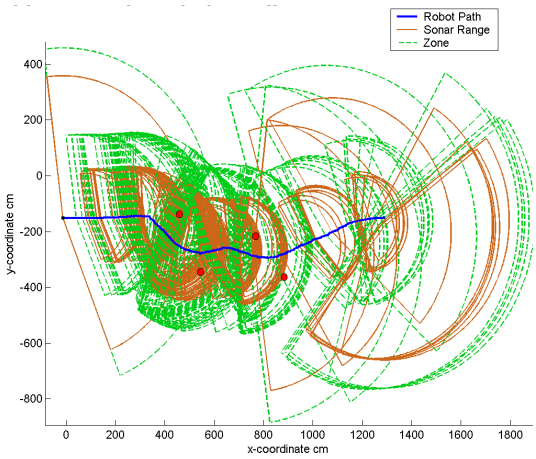


Figure 16. Zone and sonar range plotted on robot path



we see that zones become smaller as the robot gets closer to the obstacle since the robot reduces the sonar range and processing window in order to increase the robot's speed. As the robot gets closer to the end of its path, the zones go back to their initial size as the path clears from obstacles and the robot is able to adjust the processing window back to its initial size while maintaining the desired speed.

Table 3 shows a comparison between both cases in terms of total time t_{total} needed to traverse the path, average calculated speed $\bar{v}$ (calculated by processing window adjustment), average actual speed $\overline{v_{actual}}$ (measured from the motors), ratio of average calculated speed to the desired speed $\bar{v}/v_{desired}$, the ratio of the average actual speed to the desired speed $\overline{v_{actual}}/v_{desired}$, total slack time $t_{slacktotal}$ over the whole path, $\overline{t_{slack}}$ average slack time over the whole path, and $t_{slacktotal}/t_{total}$ is the ration of the total slack time to the total time needed to complete the path. These results show that the speed adjustment algorithm provided about 14% improvement relative to the desired speed. The speed improvement in the experiment was less than than the improvement in the simulation because we have accounted for the speed transition delay by adopting a linear approximation to Equation (30). The simulation result also shows that there is more slack time gained by using the processing window adjustment algorithm. However, the gain in slack time was smaller than the simulation case due to the fact the obstacles were closer to the path than the simulation and the safety margin S_M is smaller than than simulation scenario.

8. Conclusion

We presented a method for integrating the sensor and speed requirements of a mobile robotic platform with real-time fixed priority scheduling. To do this, new abstractions called zones and processing windows were created. Then a method of analyzing the processing requirements for each zone and ensuring the schedulability of real-time tasks on the platform was presented. We have shown that by adjusting sensor sampling rates the performance of the mobile robotic system can be improved in terms of maintaining a desired speed while allowing more tasks to be executed on the platform processor.

References

[1] N. Audsley, A. Burns, M. Richardson, and K. T. A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.

[2] F. Baccelli, B. Gaujal, and D. Simon. Analysis of preeptive periodic real-time systems using the (max,plus) algebra with applications in robotics. *IEEE Transactions On Control Systems Technology*, 10(3):368–380, May 2002.

[3] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems ECRTS*, pages 153–158, York, U.K., June 1999.

[4] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fact mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6):1179–1187, September-October 1989.

[5] J. J. Craig. *Introduction To Robotics: Mechnics and Control*. Prentice Hall, third edition edition, 2005.

[6] A. Das, R. Fierro, V. Kumar, B. Southall, J. Spletzer, and C. Taylor. A real-time vision-based control of a nonholonomic mobile robot. In *Proceedings of 2001 IEEE International Conference on Robotics and Automation*, pages 1714–1719, 2001.

[7] M. Dertouzos. Control robotics: The procedural control of physical processes. In *Proceedings of the IFIP Congress*, pages 807– 813, 1974.

[8] S. Farritor and M. Rentschier. Robotic highaway saftey marker. In C. Mellish, editor, *ASME International Mechanical Engineering Congress and Exposition*, Montreal, May 2002.

[9] R. George and Y. Kanayama. A rate monotonic schedular for the real-time control of autonomous robots. In *Proceedings of the 1996 IEEE International Confernce on Robotics and Automation*, Minneapolis, Minnesota, April 1996.

[10] H. Hassan, J. Simo, and A. Crespo. Enhancing the flexibility and the quality of service of autonomous mobile robotic applications. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems ECRTS*, 2002.

[11] J. Huang, S. Farritor, A. Qadi, and S. Goddard. Localization and follow-the-leader control of a heterogeneous group of mobile robots, ieee/asme transactions on mechatronics. *IEEE/ASME Transactions on Mechatronics*, 11(2):205215, March 2006.

[12] R. Kumar, B. Kimiaghalam, and A. Homaifar. Reactive real time behavior for mobile robots in unknown environments. In *Proceedings of IEEE International Symposium on Industrial Electronics*, pages 693–697, 2004.

[13] H. Li, J. Sweeney, K. Ramamritham, R. Grupen, and P. Shenoy. Real time support for mobile robotics. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 10–18, May 2003.

[14] S. Lin, G. Manimaran, and B. L. Steward. Feedback-based real-time scheduling in autonomous vehicle systems. In *Proceedings of 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 316 – 323, Tornto, Canada, May 2004.

[15] C. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[16] T. A. N. Miyata, J. Ota and H. Asama. Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment. *IEEE Transactions On Robotics and Automation*, 18(5):769–780, October 2002.

[17] M. Piaggio, A. Sgorbissa, and R. Zaccaria. Preemptive versus non-preemptive real time scheduling in intelligent mobile robotics. *Journal of Experimental and Theoretical Artificial Intelligence*, 12(2):235–245, September-October 2000.

[18] D. Prasad and A. Burns. A value-based scheduling approach for real-time autonomous vehicle control. *Robotica*, 18:273–279, 2000.

[19] A. Qadi, S. Goddard, J. Huang, and S. Farritor. A performance and schedulability analysis of an autonomous mobile robot. In *Proceedings of The 17th Euromicro Conference on Real-Time Systems*, pages 239– 248, Palma de Mallorca, Spain, July 2005.

[20] A. Qadi, S. Goddard, J. Huang, and S. Farritor. Dynamic speed and sensor rate adjustement for mobile robotic systems. In *Proceedings of The 19th Euromicro Conference on Real-Time Systems*, pages 239– 248, Pisa, Italy, July 2007.

[21] J. Shi, S. Goddard, A. Lal, and S.Farritor. A real-time model for the robotic highway safety marker system. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Application Symposium*, pages 331–440, Toronto, CA, May 2004.

[22] M. Wargui, M. Tadjine, and A. Rachid. A scheduling approach for decentralized mobile robot control. In *Proceedings of the 1997 IEEE/RSJ International Conference on system Intelligent Robots and Systems*, pages 1138–1143, September 1997.

[23] M. Zaera., M. Esteve, C. Palau, J. Guerri, F. Martineza, and P. de Cordoba. Real-time scheduling and guidance of mobile robots on factory floors using monte carlo methods under windows nt. In *Proceedings of 8th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 67–74, 2001.