# Ph.D. Dissertation Proposal

# Dynamic Processing Windows for Mobile Robotic Real Time Systems

**Ala′ Qadi**

*Department Computer Science & Engineering*

*University of Nebraska–Lincoln*

Advisor: Dr. Steve Goddard

Committee Members:

Dr. Hong Jiang

Dr. Shane Farritor

Dr. Witawas Srisa-an

Dr. Won Mee Jung

**Abstract**

*Mobile real-time systems must use sensors to scan the environment and adapt to constraints imposed by a dynamically changing environment and predictably react to those changes in real-time. Complexity arises in mobile real-time 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 dissertation is mobile real-time 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 by developing the concepts of processing windows and zones. Processing windows can become dynamic in different situations relating to the changes in the environment of a mobile real-time platform or the level of performance required from the platform. Therefore we also propose an algorithm to adjust the processing windows to meet these requirement while maintaining schedulibilty.*

## 1. Introduction

Traditional real-time systems are concerned only with guaranteeing task execution by a temporal deadline. Most of these traditional systems do not move within a dynamically changing environment where changes can affect task execution times, deadlines or the desired platform performance to meet these changes.

Mobile real-time systems must use sensors to scan the environment and adapt to constraints imposed by a dynamically changing environment and predictably react to those changes in real-time. Mobile real-time 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.

The focus of this dissertation is mobile real-time 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 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. 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 propose research for 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 remainder of this proposal is organized as follows. Section 2 presents a summary of the background, motivation and contributions of this dissertation. Section 3 presents the main concepts that form the basis for the research in this dissertation. Section 4 presents a summary of preliminary results. Section 5 discusses related work. Section 6 presents a list of items to be finished to complete the dissertation and Section 7 presents a proposed time list to finish these items.

## 2. Background and Contributions

This section provides a summary of the background, motivations and contributions of this proposed dissertation. Section 2.1 provides a background summary about real-time systems. Section 2.2 presents a summary of the contributions and motivations of this proposed dissertation.

### 2.1 Real-Time Systems

This section provides a summary of the background, motivations and contributions of this proposed dissertation. Section 2.1 provides a background summary about real-time systems. Section 2.2 presents a summary of the contributions and motivations of this proposed dissertation.

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 different real-time models let us 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 [28]. 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 $d$ :

$\phi$ 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.

However the periodic task model is not suitable for all applications because many real-time applications do not execute tasks on a periodic basis and there are periods where the processor is not executing any task. Therefore some alternate models have been suggested. The most important one of them is the sporadic task model defined by Mok [31]. Each task in a sporadic task set $\mathrm{T} = \{T_1, T_2, \ldots, T_n\}$ has three associated parameters, $p, e,$ and $d$ :

$p$ is the minimum separation 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.

Tasks can be either independent or have shared resources. Examples of shared resources are shared data in memory, printers and remote servers. Two tasks are independent if they do not share any resources. An independent task set is a task set where any two tasks in the set are independent.

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**: The 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) [28]. 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 [28]. 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.

Another important concept in real-time scheduling is preemptibility. In preemitive scheduling if a higher priority task is released then the executing task is suspended and the higher priority task is set to execute. The suspended task will continue executing if no higher priority task is available. In non-preemptive scheduling if a task is executing it will continue executing until it finishes, even if a higher priority task was released while it is executing.

## 2.2 Motivation and Contribution

In many mobile real-time systems, the system needs to maintain or maximize the performance level of the system. Examples of mobile real-time system performance can be the system velocity, accuracy of position or ability to execute extra tasks. Also the complex nature of a mobile real-time system usually involves using many sensors to collect the data and involves utilizing complex planning and control algorithms for navigation. The tasks controlling sensors and the planning algorithms are usually interdependent with a minimum compulsory delay between them. The traditional real-time systems described Section 2.1 fail to capture these aspects of mobile real-time systems. Therefore we propose the concept of processing windows to capture the special aspects of mobile-real-time systems that cannot be captured by traditional real-time systems.

The motivation behind developing the processing window concept can be summarized in the following points:

- Traditional real-time periodic model does not fit interdependent tasks with inter-delays.
- Traditional real-time periodic model does not relate the deadline of the task to any spatial concept.
- Traditional real-time periodic model does not consider dynamically changing execution requirements due to the environment.

The main contributions of this dissertation are

- an abstract analysis methodology for mobile real-time systems that integrates spatio-temporal properties using the processing windows and zone abstractions.

- an algorithm for adjusting platform speed or performance and processing windows such that the maximum speed less than or equal to the desired speed (or performance) is maintained or maximized while adjusting the processing window to maintain schedulability of the platform's real-time tasks.

## 3. Concepts

In this section we present the main concepts developed in this dissertation to help model mobile real-time systems.

### 3.1 Processing Windows and Zones

We define a *processing window* as the time interval from the instant the platform starts collecting data to the moment the platform must finish processing the data. In this context a processing window is the deadline for execution of one or more interdependent tasks. Also the processing window length can be related to more than just a pre-defined time interval, as explained in Section 3.2.

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. 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.

A mobile real-time system can be in one four states while it interacts with its environment:

- Stationary System, Static Environment
- Moving System, Static Environment
- Stationary System, Moving Environment
- Moving System, Moving Environment

#### 3.1.1 Stationary System, Static Environment

In this case the system is stationary but still needs to collect data about its environment and process the data according to time constraints. Because the system is stationary, the zone boundary is defined by the system sensors' range. Figure 1 shows an example of the zone boundary for a two dimensional zone where the system starts collecting sensor data at point $A$, and does not move while collecting the sensor data. In Figure 1 the platform uses sensors with an angle of coverage of $\theta$ and maximum range of $r$. In the two-dimensional zone shown in Figure 1, the zone is a circular section due to the sensor distribution and coverage. Therefore each zone

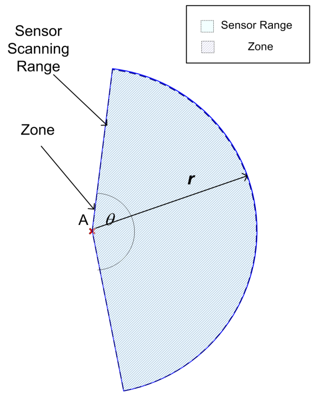


**Figure 1. Zone Boundary for Stationary System**

boundary can be defined by an angle and a radius. From Figure 1 we can see that the *zone radius* $D_i$ is equal to the sensor range $r$.

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.

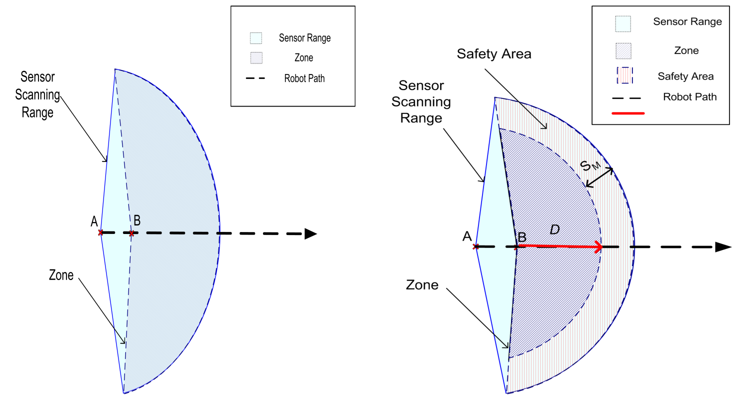
We will define the instant that the platform starts collecting data as *data collection instant* $t_i^B$ and the instant the platform must finish processing the data for the zone as $t_i^F$. In this case $W_i = [t_i^B, t_i^F)$, and the processing window duration $w$ can be calculated from Equation (1).

$$w_i = t_i^F - t_i^B \tag{1}$$

### 3.1.2 Moving System, Static Environment

In this case the system is moving while it is collecting data about its environment. Therefore other factors relating to the system's motion such as system's velocity and safe separation distance from any obstacles in the path affect the zone boundary. In Figure 2(a) 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 2(b) is further reduced because a safety area has been added for extra precautions due to sensor errors and braking distance.

In this case 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



(a) In motion

(b) In motion, safety area included

**Figure 2. Zone Abstraction: Zone Boundary**

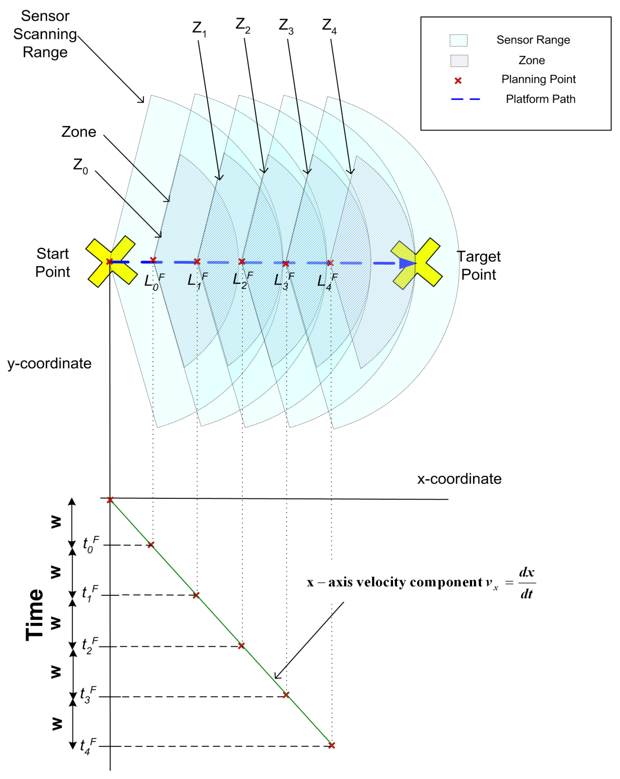
Equation (2), where $\overline{AB}$ is the distance the platform moved from point $A$ to point $B$.

$$D_i = r - \overline{AB} - S_M \tag{2}$$

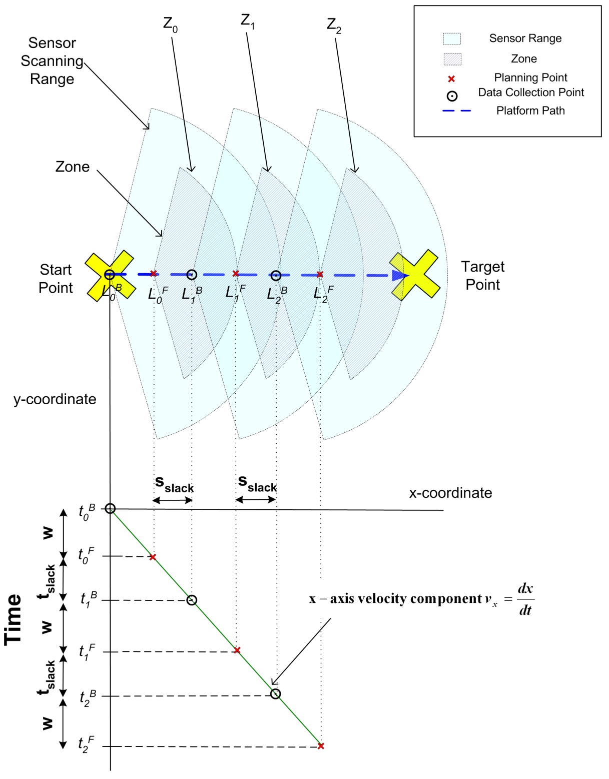
We define the point (in space and time) that the platform finishes planning its path and speed for zone $Z_i$ as *planning point* $F_i$. Because $F_i$ describes both spatial and temporal information, *planning point* $F_i$ is denoted by the tuple $(t_i^F, L_i^F)$ where $t_i^F$ represents the time instant 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. For a mobile real-time platform that moves in a two-dimensional cartesian coordinate system $L_i^F$ will be denoted by the tuple $(x_i^F, y_i^F, \psi_i^F)$ where $x_i^F$ and $y_i^F$ represent the platform's $x$ and $y$ coordinates respectively and $\psi_i^F$ 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 (3).

$$w_i = t_{i+1}^F - t_i^F \tag{3}$$

Figure 3(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 as soon as it finishes planning for the previous zone. As illustrated in Figure 3(a), the platform must start scanning zone $Z_{i+1}$ at point $F_i$.



(a) Zone concept: Maximal Scanning



(b) Zone concept: Minimal Scanning

**Figure 3. Division of robot's path into zones**

In Figure 3(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 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 $(x_i^B, y_i^B, \psi_i^B)$. 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 3(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 (4).

$$w_i = t_i^F - t_i^B \tag{4}$$

### 3.1.3 Stationary System, Moving Environment

This case is almost an inverse of the previous case. The zone boundary would be affected by the velocity at which the environment is moving, the safe separation and the distance from any objects approaching the system.

### 3.1.4 Moving System, Moving Environment

This case is more complicated and involves integrating many of the concepts presented for the previous cases. The research presented in this dissertation should provide the necessary basis required for a full modelling of this case. Therefore we leave this case in the proposed future work. Further research on this case is propped in the *To Do* list of this proposal.

## 3.2 Dynamic Processing Windows

To derive a feasible processing window, 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. In addition, under different situations the processing window's length may become dynamic. Factors that can change processing windows into dynamic processing windows can be summarized as:

- changes in the platform environment.
- increasing the maximum possible platform speed.
- increasing performance for processing window related task.

Section 3.2.1 presents an overview of schedulability impact on processing windows. Section 3.2.2 presents an overview of sensor impact on processing window length, and Section 3.2.3 presents an overview of the environmental impact on processing windows.

### 3.2.1 Schedulibilty Impact on Processing Window Length

We conjecture that any mobile real-time 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 $\mathbf{T_w}$. Each set of tasks can be scheduled using a different scheduling algorithm as long as every task in each set meets its deadline requirements. This model can be extended to include any soft real-time tasks by including them in a different lower priority task set.

### 3.2.2 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 (5) 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 $\Delta$ is the set of delays that might exist between the execution of sensor tasks in $\mathbf{T_w}$.

$$w \geq g(n, E, \Delta) \tag{5}$$

For sensors with adjustable ranges, $\Delta$ can be further divided into a set of independent delays, $\Delta I$, that must exist regardless of any other sensor parameters and a set of delays, $\Delta R$, that depend on sensor ranges limitations. Since $\Delta R$ 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 $g$.

$$w \geq g(n, E, \Delta I, R) \tag{6}$$

### 3.2.3 Environmental Impact on Processing Window Length

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 can be scanned and signals 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.

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, $v_{maxi}$, in two distinct cases: an obstacle free environment and an environment in which obstacles exist.

**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, $B_{i+1} = F_i$.

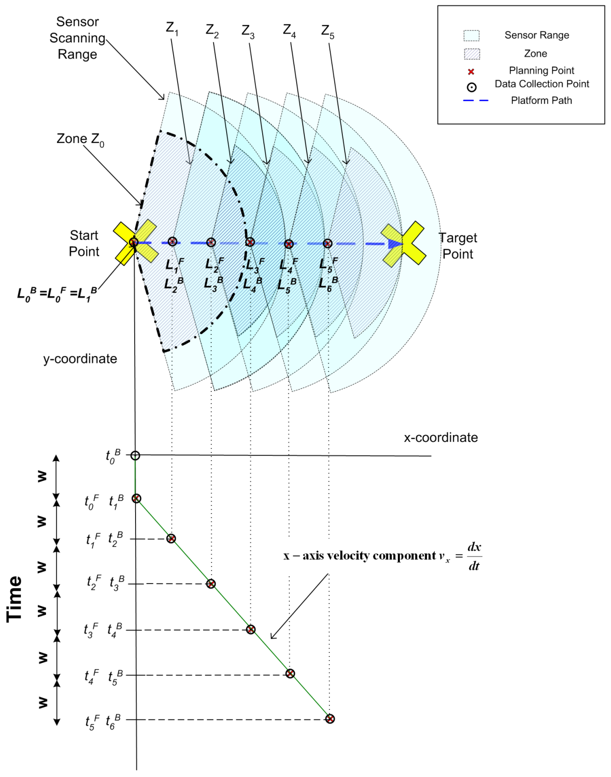


Figure 4. Maximal scanning with no obstacles

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 required for the platform to collect sensor data and plan the path for zone $Z_i$. Therefore 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 (7), where $f(v_{maxi}, v_{i-1}, w_i, D_i)$ is a formula that depends on the equation representing velocity as a function of time and displacement during the time interval the platform traverses the zone.

$$f(v_{maxi}, v_{i-1}, w_i, D_i) = 0 \tag{7}$$

Lets consider a two dimensional zone example where the platform travels at a constant speed. In this case Equation (7) simplifies to Equation (8)

$$v_{max} = \frac{D_i}{w_i} \tag{8}$$

The zone radius $D_i$ can be calculated from Equation (2). If we assume the platform is traveling at the maximum possible speed $v_{max}$ then the distance the platform moves between points $B_i$ and $F_i$ is equal to $v_{max} \cdot w$. Therefore the zone radius can be calculated from Equation (9).

$$D_i = r - v_{max} \cdot w - S_M \tag{9}$$

Substituting Equation (9) in Equation (8) and solving for $v_{max}$ we get

$$v_{max} = \frac{r - v_{max} \cdot w - S_M}{w} \tag{10}$$

$$= \frac{r - S_M}{w} - v_{max} = \frac{r - S_M}{2 \cdot w} \tag{11}$$

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 (11) becomes

$$v_{maxi} = \frac{r_i - v_{i-1} \cdot w_i - S_M}{w_{i+1}}, \tag{12}$$

where $v_{maxi+1}$ is the next maximum speed, $w_{i+1}$ is the next processing window, $w_i$ is the current sensor detection distance, $v_i$ and $w_i$ are the current speed and processing window respectively.

**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 $X_{obs} < D_i$ and we assume that the platform is moving in speed can be calculated from Equation (13), where $f(v_{maxi}, v_{i-1}, w_i, X_{obs})$ is a formula that depends on the equation describing the velocity as a function of time and displacement during the transition time interval.

$$f(v_{maxi}, v_{i-1}, w_i, X_{obs}) = 0 \tag{13}$$

$v_{maxi}$ can be calculated by solving Equation (13) for $v_{maxi}$. In an ideal case if the speed transition time is zero and the platform is moving in a constant speed through the zone, Equation (13) reduces to Equation (14)

$$v_{maxi} = \frac{X_{obs}}{w_i}. \tag{14}$$

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.

### 3.2.4 Processing Window Adjustment Algorithm

Under different situations, environmental factors might affect the processing window length, and a higher performance level (i.e., speed) might require a different processing window length (typically smaller). Because bounds on processing window length are already set due to sensor requirements, an algorithm is needed to adjust the processing window lengths to maximize performance while ensuring schedulibilty. The algorithm should adjust the processing window length by adjusting sensor or environment parameters (within a possible range). Figure 5 shows a black box diagram of such an algorithm where the inputs to the algorithm are: the schedulibilty requirements, sensor requirements and environment requirements. The outputs of the algorithm are: minimum bounds on the processing window length and an associated set of maximum bounds on system performance parameters.

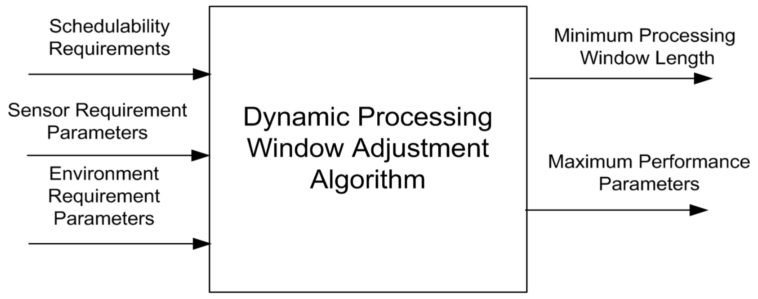

Schedulability Requirements
Sensor Requirement Parameters
Environment Requirement Parameters
Dynamic Processing Window Adjustment Algorithm
Minimum Processing Window Length
Maximum Performance Parameters

**Figure 5. Processing Window Adjustment Algorithm**

A better design approach is to sperate the system schedular from the processing window adjustment algorithm. The system scheduler schedules the tasks according to the bounds generated by the processing window adjustment algorithm. However, if the system is not schedulable under these bounds, the schedulability requirements are fed back to the processing window adjustment algorithm as shown in Figure 6.

## 4. Preliminary Results

We have tested the proposed framework of an instance of the problem on an autonomous mobile robot that was used as the lead robot in the Robotic Safety Marker project [11, 40, 36]. Through both simulation and

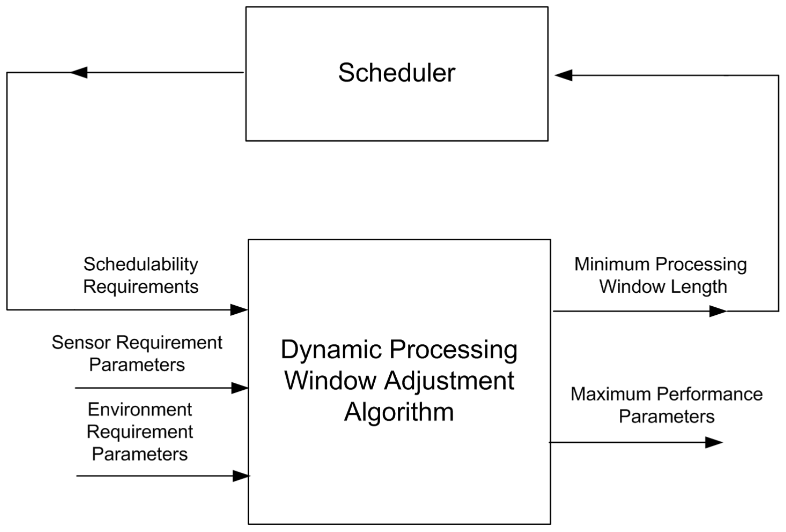

Scheduler
Schedulability Requirements
Minimum Processing Window Length
Sensor Requirement Parameters
Dynamic Processing Window Adjustment Algorithm
Environment Requirement Parameters
Maximum Performance Parameters

**Figure 6. Processing Window Adjustment Algorithm & Schedular**

real experiment, a specific instance of the algorithm described in Section 3.2.4 was developed and tested. The processing window is adjusted according to the algorithm shown in Figure 7, 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 3.2.3). The initial value is set to its desired speed as long as that value is less than or equal to $v_{max}$.

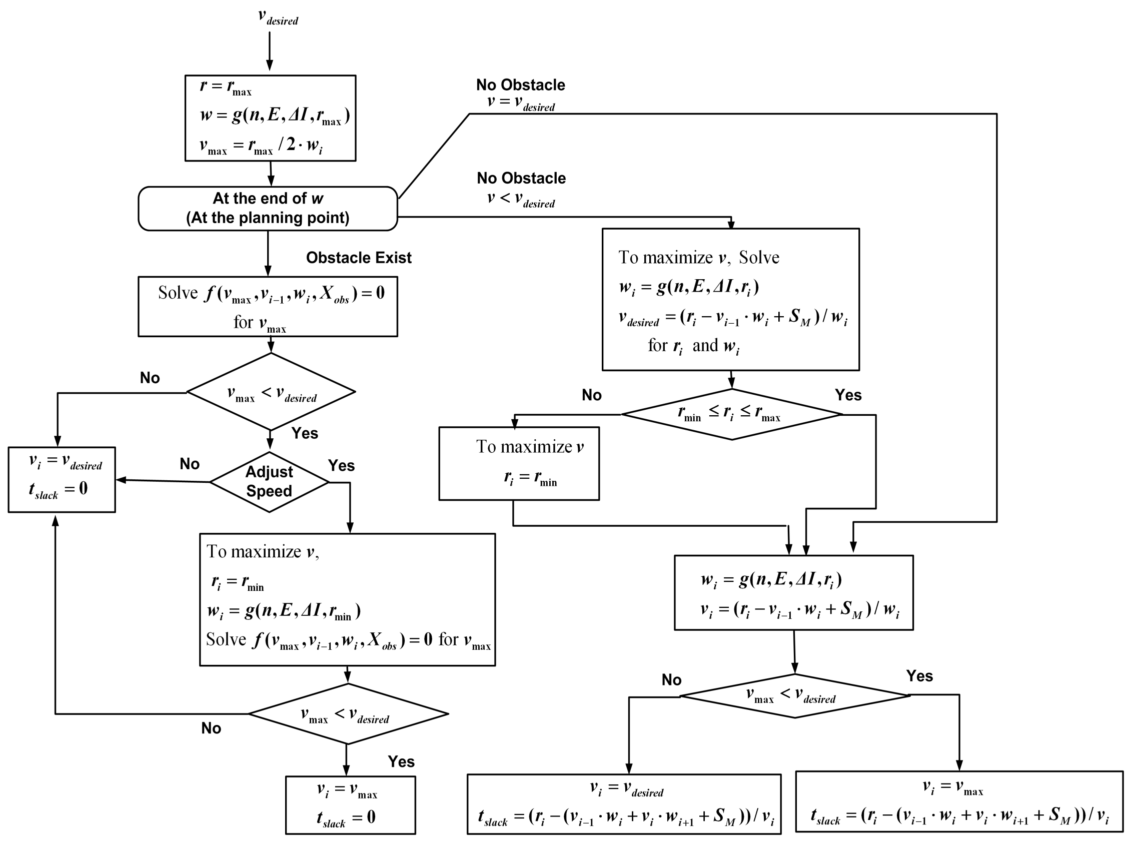
At the end of the zone processing window, there will be two cases: either there is an obstacle in the path or the path is obstacle free.

**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 adjust 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.

**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 processing 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 $w$.



$v_{desired}$
$r = r_{\max}$
$w = g(n, E, \Delta I, r_{\max})$
$v_{\max} = r_{\max} / 2 \cdot w_i$
At the end of w (At the planning point)
No Obstacle $v = v_{desired}$
No Obstacle $v < v_{desired}$
Obstacle Exist
Solve $f(v_{\max}, v_{i-1}, w_i, X_{obs}) = 0$ for $v_{\max}$
To maximize $v$, Solve $w_i = g(n, E, \Delta I, r_i)$ $v_{desired} = (r_i - v_{i-1} \cdot w_i + S_M) / w_i$ for $r_i$ and $w_i$
$v_{\max} < v_{desired}$
No
Yes
$r_{\min} \le r_i \le r_{\max}$
$v_i = v_{desired}$ $t_{slack} = 0$
Adjust Speed
To maximize $v$ $r_i = r_{\min}$
To maximize $v$, $r_i = r_{\min}$ $w_i = g(n, E, \Delta I, r_{\min})$ Solve $f(v_{\max}, v_{i-1}, w_i, X_{obs}) = 0$ for $v_{\max}$
$w_i = g(n, E, \Delta I, r_i)$ $v_i = (r_i - v_{i-1} \cdot w_i + S_M) / w_i$
$v_i = v_{\max}$ $t_{slack} = 0$
$v_i = v_{desired}$ $t_{slack} = (r_i - (v_{i-1} \cdot w_i + v_i \cdot w_{i+1} + S_M)) / v_i$
$v_i = v_{\max}$ $t_{slack} = (r_i - (v_{i-1} \cdot w_i + v_i \cdot w_{i+1} + S_M)) / v_i$

**Figure 7. Speed-processing window adjustment algorithm**

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

**Table 1. Task parameters**

## 4.1 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 ideal speed transition conditions (i.e., no speed transition time when switching between speeds). The simulation uses the parameters of an autonomous mobile robot that was used as the lead robot in the Robotic Safety Marker project [11, 40, 36]. 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

| | 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% |

**Table 2. Simulation results summary**

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. For the current sonar senor set, the feasibility function $g$ is given by Equation (15),

$$g(n, E, \Delta I, R) = n \cdot (e_{send} + e_{recv} + \tau + \frac{2 \cdot r}{340}) + e_{map} + e_{plan} \tag{15}$$

where $n$ is the number of sonar sensors used; $\tau$ 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 execution time of the map task, and $e_{plan}$ is the execution time of the plan task. 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.

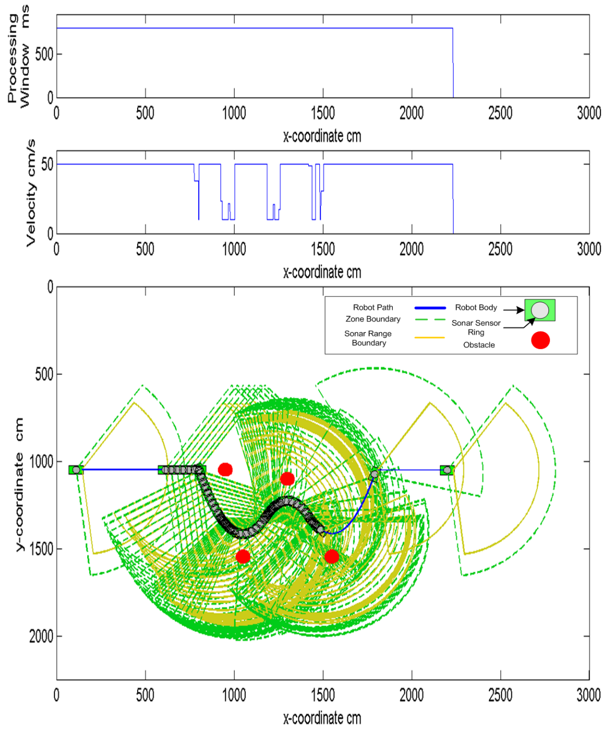
The simulation environment is event based and simulates of a 30m x 22.5m space where the robot moves. The space matrix is projected onto a visualization image where each pixel represents 1cm x 1cm of space.

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. As the platform detects an obstacle in its original path it is forced to change its path to avoid the obstacle. The robot takes a parabolic path while avoiding the 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 8(a) and 8(b) show the simulation of the robot moving along the path showing both zones and actual sonar range on the robot's path. Figure 8(a) shows the simulation of the robot traversing the path without using the speed adjustment, while Figure 8(b) shows the simulation of the robot traversing the path using the processing window adjustment. The figures show a schematic 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 8(b) 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


(a) Test with no processing window adjustments

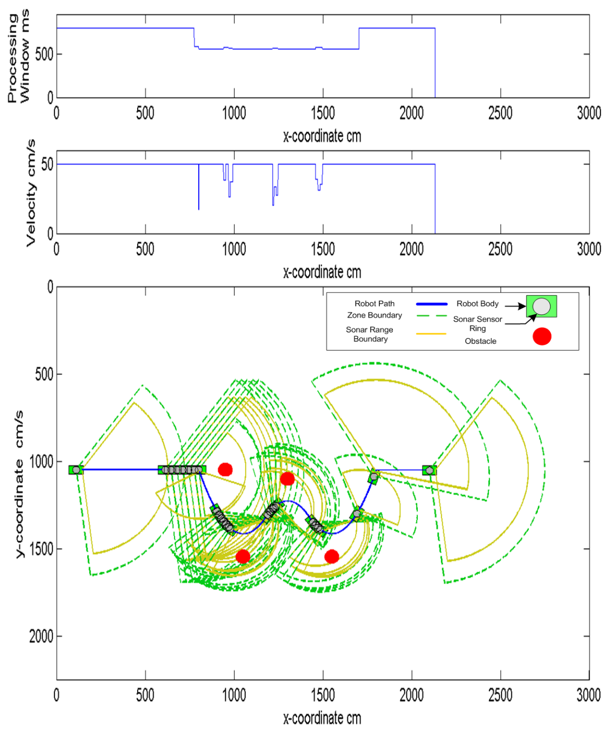

(b) Test with processing window adjustments

**Figure 8. Simulation results**

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 $\bar{v}$, and the ratio of average speed to the desired speed $\bar{v}/v_{desired}$. The result shows that the processing window adjustment algorithm improved the average speed for the robot over the whole path by about 20% relative to the desired speed.

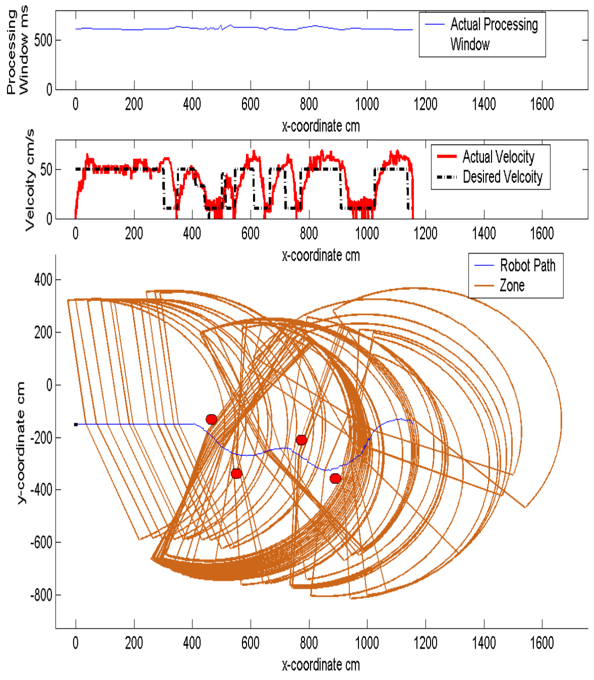
## 4.2 Experiments

We also evaluated the processing window adjustment algorithm on the actual robot described in Section 4.1. The test scenario is similar to the simulated scenario, but we have adopted a linear approximation of the speed transition function of Equation (13) given in Equation (16) where $m$ is a constant dependant on the speed controller. The derivation of the function $f$ described in detail in [37].

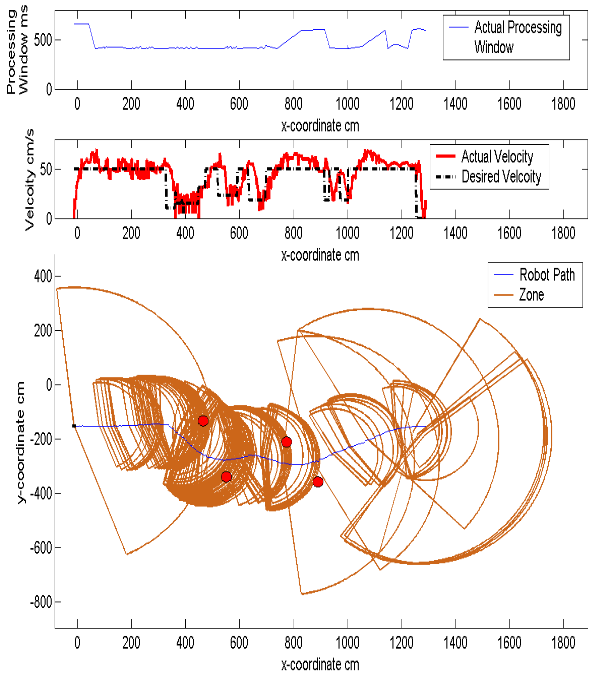
$$f(v_{maxi}, v_{i-1}, w_i, X_{obs}) = \frac{-{v_{maxi}}^2}{2m} + v_{maxi}\left(w + \frac{v_{i-1}}{m}\right) - \frac{v_{i-1}^2}{2m} - X_{obs} \qquad (16)$$

Because one of the goals of this research is the automatic adjustment of speed and processing windows for

**Figure 9. Test Progress Pictures**



(a) Zone demonstration on robot path no processing window adjustments



(b) Zone demonstration on robot path with processing window adjustments

**Figure 10. Experimental test results**

each zone, the platform was steered manually through its path, moving closer to objects than the path planning algorithm would.

We can see in Figure 10(a) 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 10(b) 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

|  | 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% |

**Table 3. Experimental results summary**

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 speed 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 $\bar{v}/v_{desired}$ and the ratio of the average actual speed to the desired speed $\overline{v_{actual}}/v_{desired}$. 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 the improvement in the simulation because we have accounted for the speed switching delay by adopting a linear approximation to Equation (13), described in detail in [37].

## 5. Related Work

In the context of mobile real-time systems, the most closely related work from the literature is based on the application of real-time scheduling theory to robots.

Applying traditional real-time systems scheduling theory to robotic applications is not a new concept. Examples of applying the classic periodic task model to robotics can be found in [26, 2, 32, 35, 14]. Examples of applying rate monotonic (RM) [28] scheduling algorithm to the control of an autonomous mobile robot can be found in [4, 14, 34, 41], the earliest deadline first (EDF) [28] scheduling algorithm can be found in [34, 42] and feedback based scheduling techniques in [17, 27, 41].

An early example of applying real-time scheduling theory to a robotic application is the work by George and Kanayama [14] in which they applied the RM [28] scheduling algorithm to the control of an autonomous mobile robot.

Prasad and Burns [35] proposed a method for ranking services on autonomous vehicle system pre-runtime. Their method assigns a value for each service based on many factors including the time 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. Their work was supported by simulation results only.

Wargui et al. [41] used real-time scheduling theory to address the communication time delay 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 authors include the delays in the derivation of scheduling bound based on rate monotonic scheduling.

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. [32] 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 in to 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. [26] presented an algorithm to distribute periodic real-time tasks on a team of mobile robots. The proposed method converts robotic applications into strategies that can be modelled with acyclic task graphs. Then they proposed an algorithm to distribute the task graphs to member robots in a team to achieve feasibility. However, their work did not consider the effect of spacial or velocity requirements of the environment on the real-time characteristics of the robot.

Zaera et al. [42] developed a real-time multithreading robotic application under the Windows NT operating system. Several mobile robots transport tiles over the factory floor using the shortest path from their current positions to the destinations. They argued that static real-time scheduling algorithms are restrictive and inflexible for many non-deterministic eal-time systems including their application. They used the EDF algorithm with PIP (Priority Inheritance Protocol).

Piaggio et al. [34] compared two different real-time scheduling policies in the navigation of an autonomous mobile robot: preemptive RM and non-preemptive EDF. From their experimental results their conclusion was that non-preemptive EDF is more efficient than RM at higher loads due to the reduction in the thread number of context switches. They also argued that non-preemptive EDF is preferable from a programming point of view because it is easier to implement.

Beccari et al. [4] presented rate modulation scheduling techniques for adaptation of soft real-time loads to available computation capacity in the context of autonomous robot control architectures. These techniques are used for overload management due to the high number of sensors and the need to perform complex reasoning activities, especially when operating in dynamic and possibly unstructured environments. Their methods are based on the knowledge of worst-case execution time of tasks and are focused on the period adjustment of soft real-time tasks within a range of admissible rates. Their work was based only the the RM scheduling algorithm. Our work considers fixed priority scheduling of tasks related to zone processing windows.

Lin et al. [27] present another feedback based real-time adaptive scheduling method for an autonomous vehicle which 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.

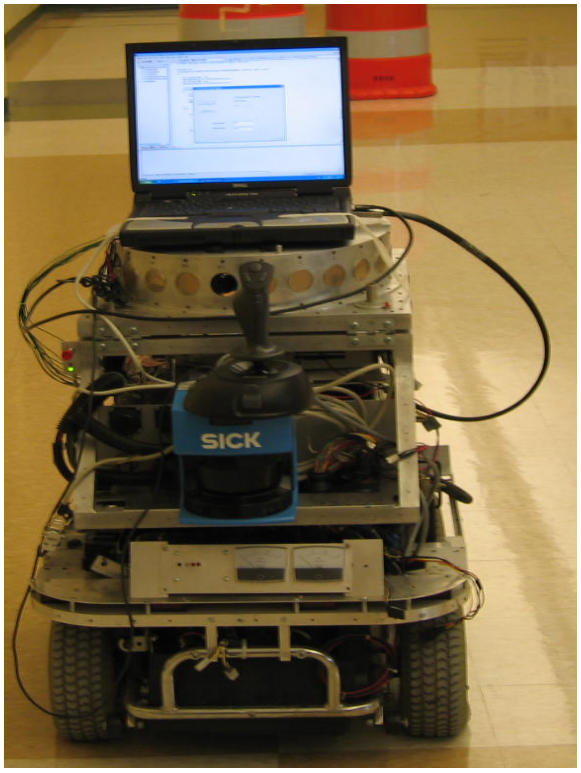
While many of the previously mentioned papers applied real time scheduling theory to a robotic application, none of these papers considered the execution requirements of the robots sensing and planning as a factor in the robot's velocity calculations. The current literature does not address the issues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

The closest work to our work is the work done by Hassan et al. [17]. 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 the velocity calculation to the robot's sensing abilities or changes in the environment. Their work and results are purely based on theoretical and simulation results without actual experiments with a robotic platform.

## 6. To Do

1. Research Case 2: Stationary System, Moving Environment.
2. Generalize the algorithm for adjusting processing windows described in Section 3.2.4.
3. Simulation and implementation
   - Simulation: A Simulation of a mobile real-time system executing different task sets moving under different environment conditions will be implemented. The simulation evaluates the the zone and processing window abstractions and the generalized processing window adjustment algorithm.
   - Implementation (Case Studies): I have worked on four robotic platform during my PhD that can serve as platforms for implementation and evaluation of the framework proposed in the dissertation. These robotic platforms are

   (a) (Companion) Lead Robot in the Robotic Highway Safety Marker Project [11, 40, 36]. Companion consists of seven units: main unit, power unit, communication unit, localization unit, sonar unit, sensor unit and motor unit. The main unit, which is the central processing unit of the foreman, consists of a PC/104-*Plus* embedded processor system running Windows CE with RS232 and RS485 serial ports


(a) Companion Robot

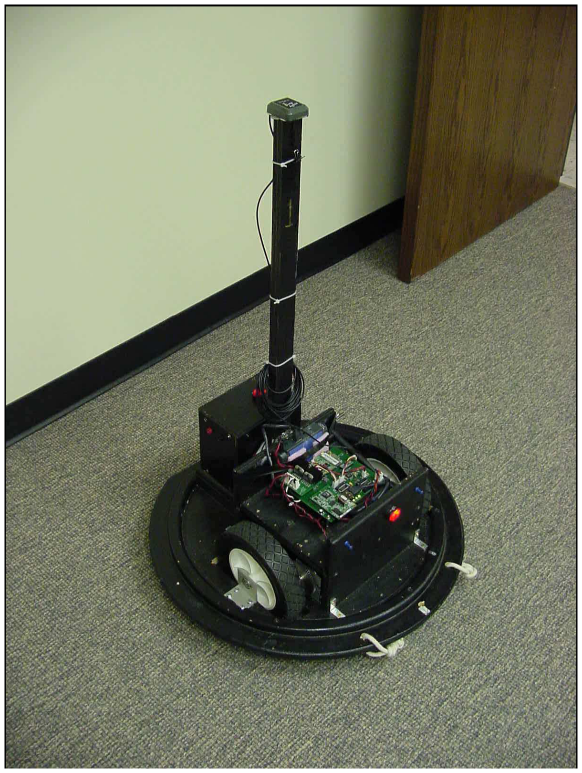

(b) RSM Robot

**Figure 11.**

and a parallel port interface. The operating system for the main unit is Windows CE. The power unit consists of two 12V batteries and DC converters. It supplies ±12V and ±5V voltages for the system. A standard RS232 serial port is used to interface with the communication unit—a 9XStream$^{TM}$ 900 MHz OEM RF module. The main control is differed from the PC/104-*Plus* to a laptop running windows XP. The robot is shown Figure 11(a).

(b) Follower Robots in the Robotic Highway Safety Marker Project (RSMs). These robots are used as the followers in the Robotic Highway Safety Marker Project [11, 40, 36]. The robotic safety barrel replaces the heavy base with a mobile robot that transports the safety barrel. These robots have a base diameter of 50 cm and 20 cm diameter wheels that are independently driven by two motors. The RSMs utilize a Rabbit 3000 processor [38] that runs MicroC-OSII [21]. The robots use communication unit—a 9XStream$^{TM}$ 900 MHz OEM RF module to communicate with leader robot. In addition the robots have a GPS unit. The robot is shown in Figure 11(b).

(c) Cliff-bots for Planetary Exploration: The Cliff-bot system consists of three individual planetary rovers that work as a team to explore the surface of a cliff [33]. Two of the rovers, designated Anchor-bots assist the motion of a third rappelling Cliff-bot down and along a cliff face using tethers. These rovers work together, as a team, in a tightly coordinated motion. The Anchor-bots use winches to control the tension in the tethers and assist the Cliff-bot with motion in any direction along the surface of the cliff or canyon wall. All of these rovers utilize a Rabbit 3000 processor [38] that runs MicroC-OSII [21]. All rovers measure tether speed using a motor encoder. The Cliff-bot is also equipped with two load cells to measure the tension in the tethers. The rovers communicate with each other and a PC through ethernet (to be upgraded to wireless ethernet). The rovers are shown in Figure 12.

(d) Crane-bots: The Crane-bot system consists of three individual rovers. Two Crane-bots and a Wall-bot. The Wall-bot is used to maneuver along the side of a building for surveillance. The Crane-bots are the same as the anchor-bots for planetary exploration but with cranes added to the rovers. The Wall-bot is a lighter robot that cannot drive on the side of the building. The Crane-bots rappel the Wall-bot on the side of the building through tethers. In this case, the rappelling robot is completely controlled by the winches on the anchor robot. The rovers are shown in Figure 13.

We already have preliminary results tested on Companion. The other platforms serve as possible candidates for implementation case studies.

4. Writing the dissertation

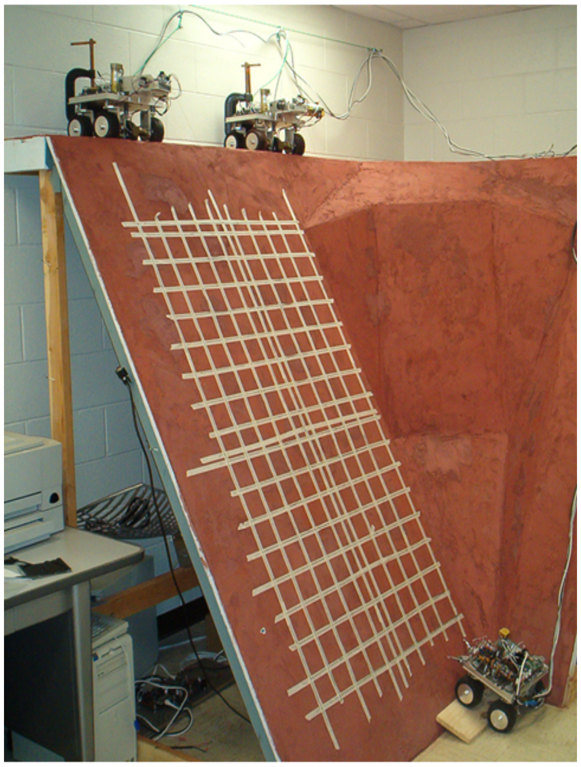
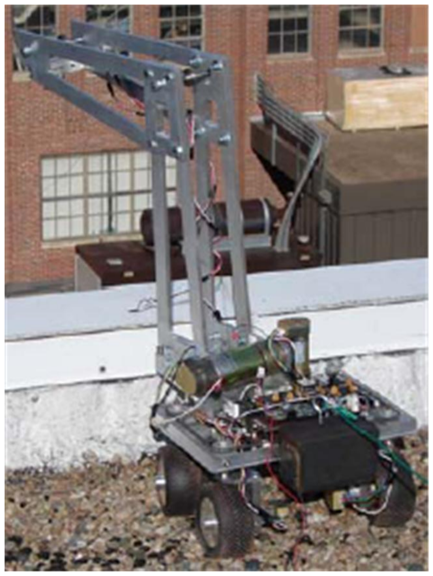
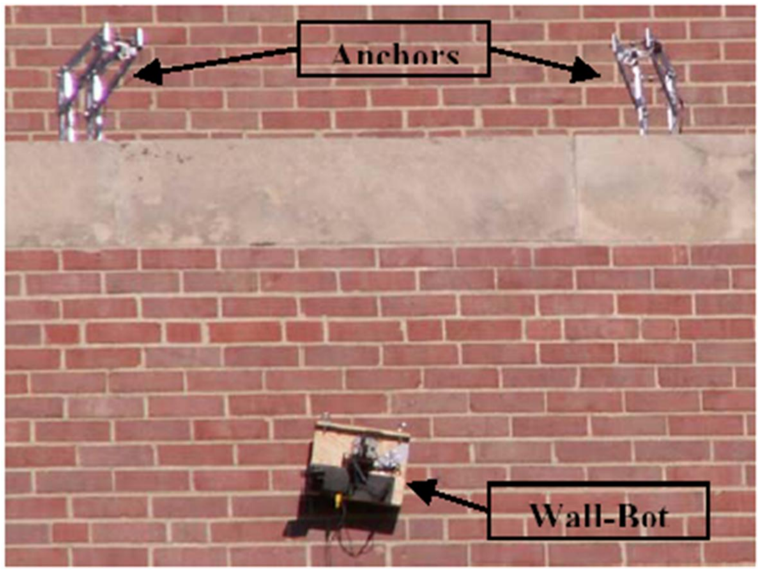


**Figure 12. Cliff-Bots Rovers for Planetary Explanation**


(a) Crane-Bot Rover



(b) Wall-Bot System Setup

**Figure 13. Crane-Bot Rovers**

## 7. Time Table

1. Research Case 2: Stationary System, Moving Environment (3-4)Weeks.
2. Develop an algorithm(s) for adjusting processing windows (4-5) Weeks.
3. Simulation and implementation (Hard to tell, depends on 1)
4. First dissertation draft (6-8 Weeks)
5. Graduate by December 2007

## 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] T. Balch and R. Arkin. Behavior-based formation control for multirobot teams. *, IEEE Transactions on Robotics and Automation*, 14(6):926939, December 1998.

[4] 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.

[5] E. Bini and M. D. Natale. Optimal task rate selectionin fixed priority systems. In *Proceedings of 10th IEEE Real-Time Systems Symposium RTSS*, pages 399 – 409, Miami, Fl, December 2005.

[6] 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.

[7] A. Burns, D. Prasad, A. Bondavalli, F. D. Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46:305–325, 2000.

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

[9] 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.

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

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

[12] M. Felder and M. Pezze. A formal design notation for real-time systems. *ACM Transactions on Software Engineering and Methodology*, 11(2):149–190, April 2002.

[13] P. Fiorini, S. Hayati, M. Heverly, and J. Gensler. A hopping robot for planetary exploration. In *Proceedings of the IEEE Aerospace Conference*, pages 153–158, March 1999.

[14] 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.

[15] S. Goddard. *On the Management of Latency in the Synthesis of Real-Time Signal Processing Systems from Processing Graphs*. PhD thesis, University of North Carolina at Chapel Hill, Department of Computer Science, 1998.

[16] S. Goddard and K. Jeffay. Analyzing the real-time properties of a dataflow execution paradigm using a synthetic aperture radar application. In *Proceedings of the 3rd IEEE Real-Time Technology and Application Symposium*, pages 60–71, Montreal,CA, June 1997.

[17] 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.

[18] 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.

[19] 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.

[20] Y. Kwok and I. Ahmed. Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, 47(1):58–77, November 1997.

[21] J. Labrosse. *The Real Time Kernel MicroC/OS-II*. CMP Books, 2002.

[22] C. Y. Lee, H. Gonzalez-Banos, and J. Latombe. Real time tracking of an unpredictable target amidst unknown obstacles. In *Proceedings of the 7th International Conference on Control, Automation and Vision (ICARCV'02)*, Singapore, December 2002.

[23] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 201–209, Lake Buena Vista, Florida, USA, December 1990.

[24] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm - exact characterization and average case behaviour. In *Proceedings of 10th IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.

[25] J.-T. Leung and J. Whitehead. On the complexity of fixedpriority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237250, December 1982.

[26] 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.

[27] 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.

[28] 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.

[29] J. Liu. *Real-time Systems*. Prentice-Hall, 2000.

[30] C. McElhone and A. Burns. Scheduling optional computations for adaptive real-time systems. *Journal of Systems Architectures*, 46:46–77, 2000.

[31] A. Mok and S. Sutanthavibul. Modeling and scheduling of dataflow real-time systems. In *Proceedings of 10th IEEE Real-Time Systems Symposium RTSS*, pages 399 – 409, Miami, Fl, December 2005.

[32] 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.

[33] G. Paulsen, S. Farritor, T. Huntsberger, and H. Aghazarian. All terrain exploration with the cliff-bot system. In *Proceedings of the ICRA 2005 Conference*, Barcelona, Spain, 2005.

[34] 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.

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

[36] 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, Miami, Fl, July 2005.

[37] A. Qadi, S. Goddard, J. Huang, and S. Farritor. Deriving speed limits and sensor sampling rates for mobile cyber-physical systems. Technical Report TR-UNL-CSE-2006-0015, Departement of Computer Science and Engineering, University of Nebraska-Lincoln, http://cse.unl.edu/ goddard/Papers/TechReports/TR-UNL-CSE-2006-0015.pdf, October 2006.

[38] R. Semiconductors. *Rabbit 3000 Microprocesser User's Manual*.

[39] X. Shen. Control of robotic highway saftey markers. Master's thesis, Mechanical Engineering, University Of Nebraska-Lincoln, 2003.

[40] 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.

[41] 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.

[42] 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.