

# Challenge Three:

## *Using the PPRK For Landmark Navigation*

Makiko Kaijima  
Cy Routh  
Stephen Upchurch

### ***Introduction***

This challenge involves the task of mapping objects encountered in an environment and then navigating to any selected object. The robot's "world" is a rectangle approximately 4x8 feet in dimension surrounded by a wall about 4 inches high. Inside this wall objects, such as soda bottles, are randomly placed. Ideally, mapping and navigation should be carried out by the use of simple behavior-based landmarks as opposed to the making of a detailed world-map and deliberative planning.

The design approach taken uses the Palm Pilot Robot Kit (PPRK): a small easy-to-build robot kit. The kit was developed by Carnegie Mellon University Robotics Institute. The palm pilot kit packs a lot of computing power in a small size and allows for the construction and implementation of a fully autonomous robot.

The kit has a hexagonal base with three "omni-wheels" that allow the robot to traverse in any direction with independent control of its rotation, holonomic motion. The base has three infrared range sensors. The compiler used is Metrowerks' CodeWarrior Development Studio for Palm OS Platform. This is a 'C' based programming IDE and allows for the development of programs including the interface that is displayed on the screen.

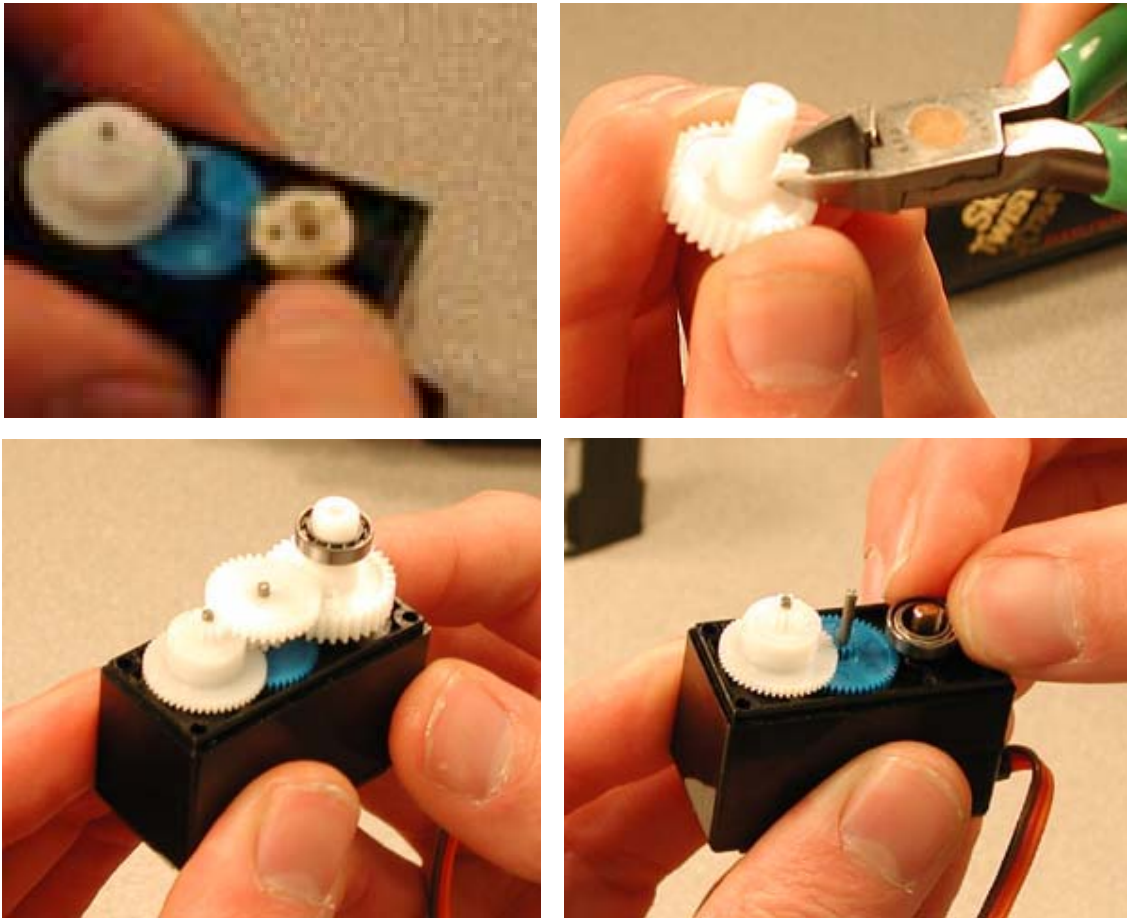
The implementation of the task assumes that the PPRK starts on a short wall and then follows the wall in a clockwise direction. As the robot follows the path along the wall it maps its environment by detecting landmarks, namely objects and corners, that can later be used for navigation. A landmark is defined by the order in which it is encountered (the landmark ID) and the type of landmark (Start, Object, or Corner). For objects, the distance to the object and an object ID are also recorded. Simple navigation can be done using just the landmark ID and type. Object distance information can be used to determine a more efficient path to a chosen object. Given that in this environment all objects will be visible by a robot that follows the walls, object IDs are implicit in the ordered and typed landmark data. However, making them explicit is necessary in order to create a representation for the user to be able to select a target object for navigation.

## ***1 – Hardware Implementation***

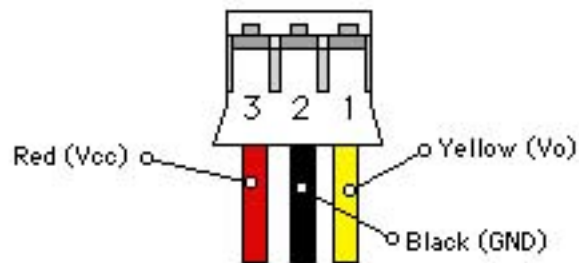
The robot uses the standard CMU PPRK; no sensors apart from the three infrared range finders are utilized. Three servos drive the three “omni-wheels”, providing holonomic, i.e. omni-directional, locomotion. The palm sits on top of the base, controlling the servos and interpreting the information from the sensors.

The PPRK kit comes with “some assembly required.” The kit was assembled, following the instructions on the Acroname, Inc. website. The servos must be “fixed”: the stop on the large gear must be removed to allow for continuous 360° rotation, and the potentiometer must be disengaged from the feedback loop of the servo. The potentiometer is connected to the small brass pin and gives the servo feedback by the little white piece of plastic. Bearings are also inserted to take the side loads that the

PPRK imposes, improve the efficiency, decrease the noise, and lengthen the life of the servo. Figure 1 shows the modifications being made to the servos. The IR sensors have a keyed plastic connector that is connected as shown in Figure 2.

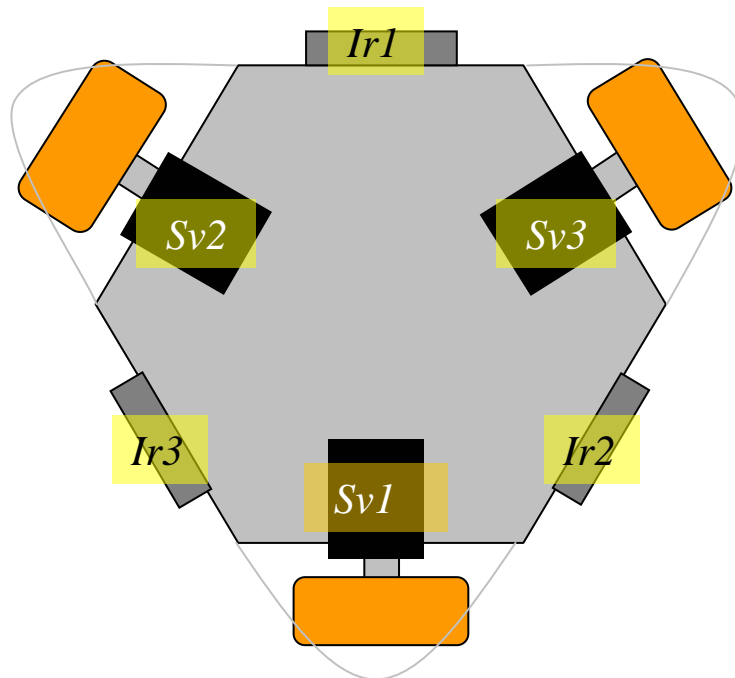


**Figure 1:** Potentiometer feedback plastic clip (top left), the fixing of the large gear (top right), and bearing installation (bottom left and right).

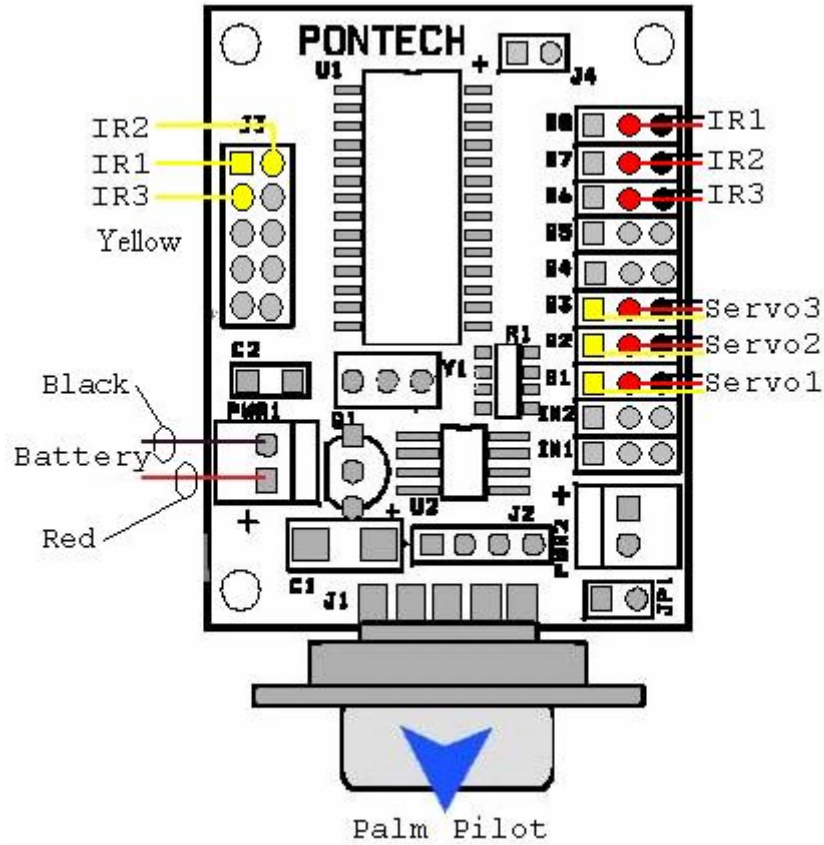


**Figure 2.** IR range finder wiring schematic.

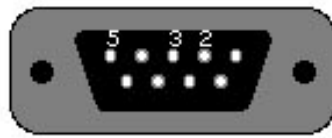
The IR sensors and fixed servos are mounted to the frame and labeled as depicted in Figure 3. The IR sensors are labeled Ir1, Ir2, and Ir3. The servos are given the same number as the opposing sensor, i.e. Sv1, Sv2, and Sv3. The Pontech SV203 controller is mounted to the clear plexiglas top, and the servos and infrared sensors are connected to the SV203 as shown in Figure 4.



**Figure 3:** Base assembly and component labeling (top view).



**Figure 4:** Servo, IR sensor, and power connections to the Pontech SV203.

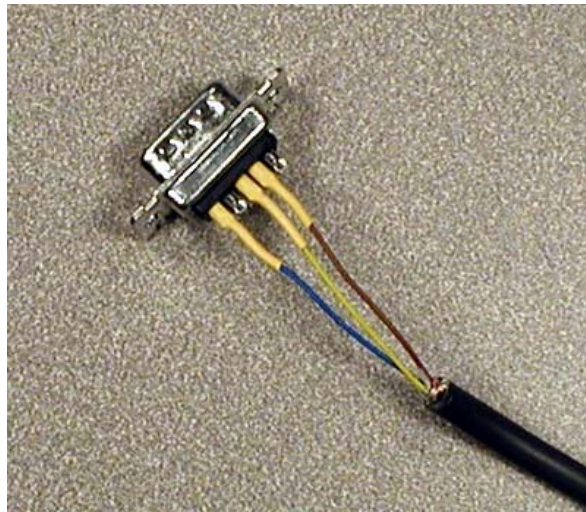


Back of DB9 Connector Shell

	Acroname PalmIII Kit Cable	Acroname PalmV Kit Cable	Hacked PalmIII Hotsync Cable
Pin 2	Brown	Green	Brown
Pin 3	Yellow	Purple	Black
Pin 5	Blue	Black	Red

**Figure 5:** Connections from the Palm cable to the pins on the serial connector.

The Pontech SV203 has a serial connector that allows for communication to and from the Palm Pilot. The cable has a Palm connector on one end and needs a serial connector attached to the other. Figure 5 details which wires are connected to which pins according to the type of Palm that is used, in this case a PalmVx. The wires are soldered to the pins on the serial connector and shrink wrapped as shown in Figure 6. Once the connector has been made, it is attached to the SV203 and fed through the top plate where it can be attached to the Palm Pilot.



**Figure 6:** Palm Connector soldered to the DB-9 serial connector.

## ***2 – Software Implementation***

To start with, we referenced the PPRK code written by Greg Reshko of Carnegie Mellon University (<http://www-2.cs.cmu.edu/~pprk>). We had difficulties compiling all the files associated with *sample.cpp*. (*sv203.h*, *sv203.cpp*, *robot.h*, *robot.cpp*, and

*vecmath.h*). The two main errors were that the compiler couldn't find the header files and math libraries of CodeWarrior had deficit.

Initially, we couldn't compile all the files; however, we managed to compile all of them except *sample.cpp* and *robot.cpp* by changing the header files in the Palm OS SDK, changing type definitions, and downloading new math libraries from <http://www.radiks.net/~rhuebner/mathlib.html>. Later, we found more information about how to convert old files to ones that can run in Palm OS version 3.5. Following the explanation below, we could have compiled all of the sample code.

### *Moving Applications to Palm OS 3.5*

With the release of Palm OS version 3.5, there were significant changes. Prior to OS 3.5, the basic headers to be incorporated for starters was *Pilot.h*. Instead in 3.5, *PalmOS.h* has to be incorporated. Also other header files and type definitions associated with the new starter header file had to be changed accordingly. For example, *Common.h* is renamed as *PalmTypes.h*. *SysAll.h* is not supported in 3.5 and you need to comment it out. The type definitions of OS 3.5 are clearer and more consistent. Also there are some header files that are new to OS 3.5; especially the inclusion of *SerialMgrOld.h*. For more complete information about the header file changes, please refer to Appendix A.

If you are working with code originally written using headers earlier than those included with Palm OS SDK 3.5, you will need to search for the older data types and replace them with their new equivalents. Much of this can be automated by including a file called *PalmCompatibility.h*, found in the 3.5 SDK, which maps the older style data types to their new names. You can include *PalmCompatibility.h* in an older project to

help it deal with compilation under the 3.5 headers, but it is probably better off in the long run to replace the data types yourself. Some tips are given in Appendix B.

### *Cooperative Movement*

The robot utilizes a behavior-based locomotion scheme in which multiple movement requests are cooperatively aggregated. A number of steps take place between the top-level `SetMove` function and the actual rotation of the servos. Robot locomotion behaviors are combinations of `SetMove` functions, each of which indicating a direction and speed for the robot. The `SetMove` function decomposes the overall direction into base rotation rates for the three servos and sends them, along with the speed, to the `SetServoDirection` function. The `SetServoDirection` function aggregates the rotations assigned to the individual servos.

Once all the components of the robots desired behavior have been set via the `SetMove` function, the `Move` command is issued. The `Move` command takes the mean of the rotations aggregated by the `SetServoDirection` function and sends three calls to the `Servo` command, one for each of the servo-mounted omni-wheels. The `Servo` command converts the mean rotation speed from radians per second to a servo position and sends an ASCII command to the Pontech SV203 controller which actually rotates the servo.

Ultimately, the wheels are rotated by sending ASCII commands to the Pontech SV203 controller. The command takes two arguments: one indicating which servo is to be rotated and the other indicating the speed at which it will rotate. The speed variable accepted by the controller is nothing more than a servo position in the range [0-255]. The actual rotational velocity of the wheel is a non-linear function of the servo position. The

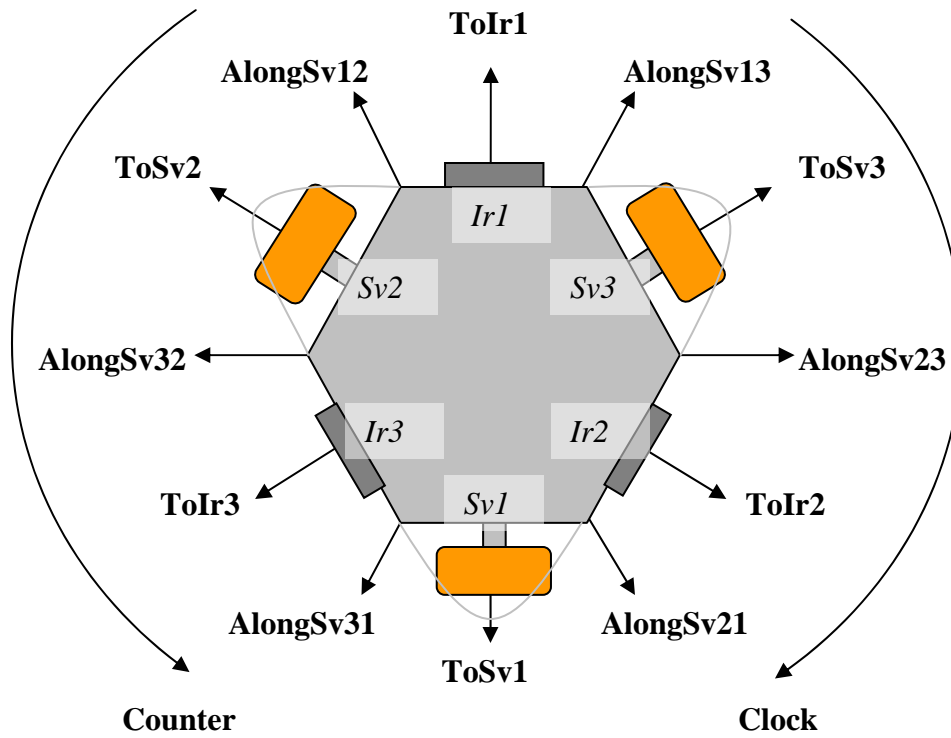
servo position as a function of the wheels velocity in revolutions per second is well approximated by the cubic regression

$$p(x) = 39.8053x^3 - 12.6083x^2 + 22.1197x + 128.4262$$

where  $p$  is the servo position and  $x$  is the velocity in revolutions per second (restricted to  $[-1,1]$ ). To simplify movement calculations, when a move is set by the `SetMove` function, it is given in radians per second. This speed is converted to revolutions per second for use with the above equation. Additionally, if a servo is set to stop, rather than convert a speed of  $0$  to a servo position, it is explicitly set to position  $0$ . This actually stops the servo, whereas using the calculated servo position leads to a slight creep in the servo.

The robot can be instructed to carry out 14 distinct movements: two movements cover rotating clockwise and counter-clockwise; six movements cover the motions in the directions of each of the three servos and the three IR sensors (or, alternatively, to and from each servo); six movements cover the motions along the lines connecting the servos, e.g. from Sv1 to Sv2, from Sv2 to Sv3, from Sv2 to Sv1, etc. The 14 movements are depicted in Figure 7. Additionally, each motion is given a speed in radians per second. For example, the function `SetMove(ToSv1,2.5)` directs the robot to move in the direction of Sv1 at a speed of 2.5 radians per second. The assignment of linguistic values to each of the 14 movements makes programming the robot extremely straightforward.

The overall movements are obtained by multiplying a modifier for each servo that sets the servo's direction and base rotation rate by the desired speed. For example, the two rotations simply turn all servos in the same direction, positive or negative. The modifiers for the three servos for each of the 14 movements are shown in Table 1.



**Figure 7:** The 14 movement directions and their linguistic representations.

	<i>Sv1</i>	<i>Sv2</i>	<i>Sv3</i>
Clock	1	1	1
Counter	-1	-1	-1
ToSv1	0	-1	1
ToSv2	1	0	-1
ToSv3	-1	1	0
ToIr1	0	1	-1
ToIr2	-1	0	1
ToIr3	1	-1	0
AlongSv12	1	1	-2.5
AlongSv13	-1	2.5	-1
AlongSv21	-1	-1	2.5
AlongSv23	-2.5	1	1
AlongSv31	1	-2.5	1
AlongSv32	2.5	-1	-1

**Table 1:** Servo rotation rate decomposition for the 14 movement directions.

Movements towards the servos and IR sensors are simple to obtain: the (corresponding) servo indicated by the movement direction is not rotated while the other two servos are rotated in opposite directions. The result is relatively reliable straight-line movement. Since the two operational servos are rotating in opposite directions, no net angular momentum results and the forces that would carry the robot perpendicular to its intended path are equal and opposite.

The situation is not so simple for the remaining six movements: the two servos indicated by the movement direction are rotated in the same directions while the third servo is rotated in the opposite direction. This should be a purely translational movement, so the angular momentums produced by the wheels should collectively cancel. Hence, a first guess would be to double the base movement factor for the third servo. In practice, this still produces a small rotational divergence towards the third servo. To compensate, we increase the speed of the third servo by a small amount. The overall result is relatively straight movement.

Given that the PPRK's wheels cannot have high traction in order to achieve holonomic motion, none of these motions (except perhaps the rotations) are reliable enough to operate under a dead-reckoning scheme. Only on an ideal surface (paper seems to work quite well) will there be the correct balance of traction and slippage to yield the intended motion for extended periods of time. Fortunately, the PPRK's holonomic motion capabilities offer a rich basis from which to implement a behavior-based movement scheme that corrects itself by utilizing feedback from the environment.

The robot utilizes just such a behavior-based movement scheme in which multiple movement requests are cooperatively aggregated. Individual movement commands are set by the `SetMove` function and stored by the `SetServoDirection` function; commands to the individual servos are summed and a running total of the number of commands recorded. When the `Move` function is called, the servos are instructed to carry out the average of all the commands for that servo and the movement system is reset, ready to accept new movement commands.

### *Infrared Distance Sensing*

Sharp GP2D12 infrared sensors are used to measure distance between the robot and objects. The accuracy of the measurements is approximately 1cm. The SV203 converts the analog sensor outputs into an integer value ranging from 0 to 255. The relation between sensor value and the distance to an object is not linear. Distance is calculated by

$$d(s) = 2141.72055s^{-1.078867}$$

where  $d$  is distance in cm and  $s$  is sensor's value from 0 to 255. Also we found that the distance function, suggested by CMU, does not make sense when the distance calculated is below 10 cm or above 80 cm. Hence if the value returned by function is too small or too big, the value of 10 and 100 is assigned respectively in our program.

Since the sensors are infrared (IR), they are not insensitive to ambient infrared radiation. The objects and wall being made out of different materials vary in their ability to reflect infrared light, and hence affect the sensor reading. Occasionally, it appears as if

the metal hinges that hold the wooden walls together absorb the infrared radiation, often producing a maximal sensor reading despite being in close proximity. Also, since the robot is continuously moving, the sensor readings fluctuate.

When a request is made for new sensor readings by calling the AllSensors function, a moving average of the IR readings is automatically produced. The moving average takes the mean of the last few sensor readings. When a new sensor reading is taken, it replaces the oldest value and a new moving average is computed. The last three Ir2 and Ir3 sensor readings, used to measure the distance to the walls, are included in this moving average. No averaging is done for Ir1 used to detect objects. This allows for smoother movement by averaging out fluctuations in the IR sensors. In addition, it was noticed that Ir2 was consistently giving distance readings that were low by 5-10cm. To correct for this we automatically add 7cm to all Ir2 readings.

### *High-Level Locomotion Behaviors*

Three high-level locomotion behaviors are required to systematically travel inside the robot's environment GoToWall, FollowWall, and TurnCorner. In the mapping stage of the task, the PPRK is designed to continually follow the wall in a clockwise manner. In order to follow the wall, the wall must first be found and the PPRK aligned with it, and when a corner is encountered it must be tightly navigated in order to maintain reliable mapping data.

The PPRK starts off by executing the GoToWall behavior in order to find the nearest wall, assumed to be a short wall. GoToWall uses distance readings from Ir2 and Ir3 and 8 basic locomotion behaviors to align with the wall. It executes repeatedly until

the sensor read within the preset wall following distance of 30cm, using a generous tolerance of  $\pm 6$ cm.

For both Ir2 and Ir3, if the sensor reads outside the distance, then the robot is set to move towards that sensor. If it is inside the distance, the robot moves away from the sensor, towards the corresponding servo. In addition, Ir2 and Ir3 readings are coupled by rotational movements that aim to equalize the two sensor readings, and hence keep the robot aligned to the wall while it comes to within the wall following distance. On top of these motions is a constant movement towards Sv1. If both of the sensors read 100cm, there is not an object or wall within range. When that is the case, the robot will rotate clockwise in attempt to locate a wall. This rotation is at a very high speed, easily overpowering the other movements in the aggregation.

Once the PPRK finds a wall, it can then proceed along the wall using the FollowWall behavior. Again, the behavior continually requests readings from Ir2 and Ir3 to stay aligned and the proper distance from the wall. It uses 10 basic locomotion behaviors and the same preset wall distance as the GoToWall behavior, but the tolerance is reduced to  $\pm 2$ cm in this case.

Whenever an IR sensor is too close to the wall, the PPRK will move away from the wall at a speed proportionally to the difference and likewise in the opposite direction if an IR is too far from the wall. Again, it couples the sensor readings by continually attempting to keep both Ir2 and Ir3 at about the same distance from the wall. If the readings are different, it will compensate by rotating the body and bringing the IRs back into alignment with the wall. If both IR sensors are less than the tolerance, the PPRK

will back away from the wall. Likewise, if both IR sensors are too far from the wall, it will move towards the wall.

For all of these base motion behaviors, the movements away from the wall are given greater speeds than those towards the wall. If the robot gets too close to the wall, within the 10cm lower bound of the GP2D12 IR sensors, the distance readings will go up rather than down. This creates a positive feedback loop that drives the robot further into the wall rather than away from it.

Finally, motions are added to move the robot along the wall. The direction is set externally, either clockwise or counter-clockwise, in anticipation of navigational wall following after the map is made. The motion along the wall is always included at a speed comparable to the other motions. Additionally, if both of the IR sensors are within tolerance, a second low speed “boost” along the wall is added.

In the initial mapping task, the robot maneuvers along the wall. However, the environment is not just a straight shot but rather a box with corners. Although the FollowWall behavior will successfully circumnavigate the area, mapping efficiency is greatly increased if the corners are detected and tightly negotiated. As the PPRK approaches a corner, the leading IR will notice a significant drop in distance. Corner detection is covered in detail below.

When a corner is detected, the PPRK initiates the TurnCorner behavior, which rotates the robot until it is aligned with the new wall. The TurnCorner behavior only uses distance readings from the leading IR sensor and includes 4 basic movement behaviors. The robot rotates at a high speed and continues in the wall following direction at a lower speed. Again, the robot’s direction, which determines the leading sensor, the rotation

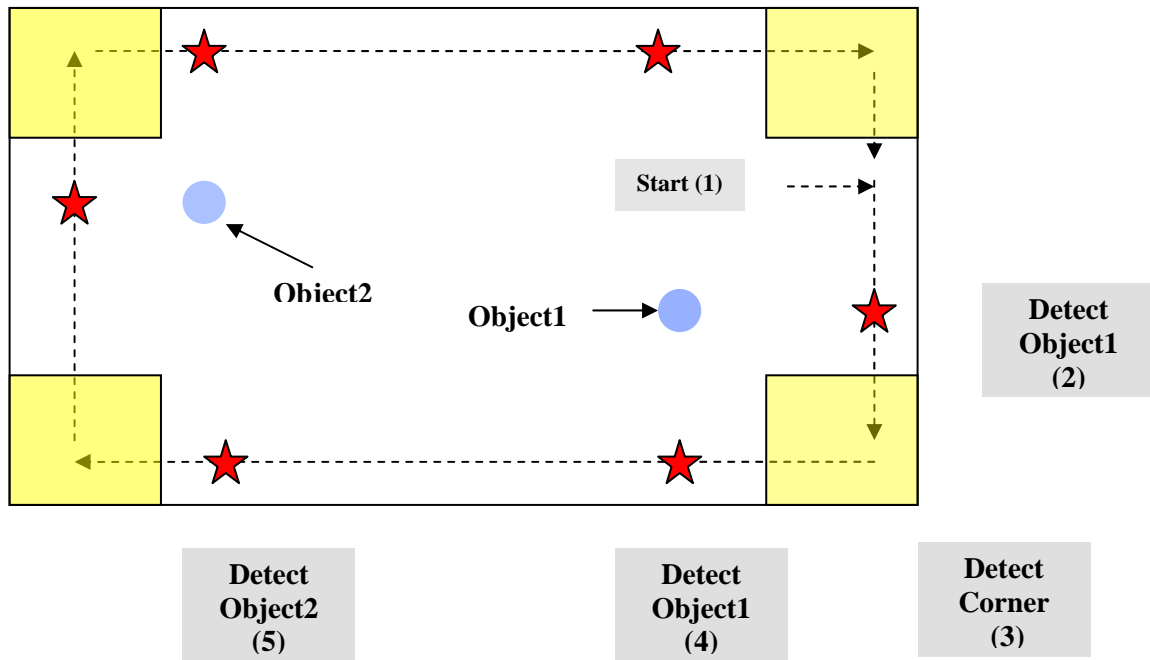
direction, and the wall following direction, is set externally. Additionally, if the leading sensor gets too close to the wall it moves away from it, and if it gets too far, it moves closer. As usual, the speed of these conditional movements is proportional to the deviation from the desired distance.

### *Landmark Detection*

Landmark information, such as type (Start, Corner, or Object), landmark ID, object ID, and the distance from the robot to an object, is stored as the robot follows the wall. See, for example, Table 2. Figure 8 depicts an example of landmark mapping. Arrows indicates the path followed by the robot. Each gray box corresponds to a landmark and number in the prentices corresponds to a landmark ID. At the point indicated by a star, an object is detected. While turning corners, indicated by the yellow turning zones, no object detection is done.

Landmark ID	Landmark Types	Object ID	Distance
1	Start	-	-
2	Object	1	30
3	Corner	-	-
4	Object	1	40
5	Object	2	40
:			
:			

**Table 2:** Example landmark data.



**Figure 8:** Sample landmark mapping.

The most difficult aspect of landmark mapping is determining the cases where different landmarks refer to the same object in the world. In an initial attempt to solve this problem two simple heuristics are used:

**Heuristic 1:** If an object is detected and the landmark type detected previously was Start or Corner, then it is the same object as the one most recently detected.

**Heuristic 2:** If an object is detected and the landmark type detected previously was Object, then it is a new object.

To execute the heuristics properly, we needed to pay special attention to detecting objects and corner regions.

The DetectObject function is continually called while the robot is following the wall. Ir1 can detect the opposite wall when the robot is on the long wall. On the other hand, when it is on the short wall, the sensor does not detect the opposite wall. Hence, two different distance tolerances are set based on the environmental knowledge. The distance reading of Ir1 is compared with the tolerance. If the result drops lower than the tolerance, the robot detects the object and updates the landmark structure. A 5 second timer is started when the object is detected to keep from detecting the same object multiple times on a single pass. This delay period was set based on the time that it would take for the robot to travel the distance between two objects that the robot could potentially navigate between.

During a turn, the distance between Ir1 and an object can drop and fall within the tolerance. Therefore, we needed to make sure that the robot does not detect any object during the turn to execute the heuristics described above correctly. Hence the corner region indicated by yellow area in Figure 8 corresponds to the period between the leading sensor's value dropping below the turning tolerance of 17cm and coming back up to the wall following tolerance of 30cm. The DetectCorner function initiates TurnCorner behavior and inhibits wall following and object detection until the corner is completely navigated.

All the information collected by the robot, i.e. sensor readings and landmark information is displayed as shown in Figure 9.



**Figure 9:** The user interface.

### ***3 – Performance and Possible Future Work***

The robot reliably circumnavigates and maps its environment. The only critical errors, occurring only occasionally, are when the robot comes too close to the wall while wall following and mistakenly detects a corner. When this happens, the robot will regularly detect the far wall as an object while it is following the long wall. Most of the programming effort centered on setting the wall following parameters, the speeds of the 10 base movements included in the behavior, to obtain smooth and reliable wall following.

The robot is not yet able to navigate to a selected object, and the ‘on-the-fly’ heuristic object identification algorithm still has some bugs. Once the object identification algorithm is debugged, it should be a relatively straightforward procedure to search the landmark map to find the shortest route to the selected object: count the number of landmarks to the desired object, counting both forward from the beginning of the landmark list and backward from the end of the list. If one of the counts yields a smaller number of landmarks, follow that route; if the counts are equal, use the distance reading for the object to break the tie.

The heuristic object identification algorithm can also be improved. Consider a case where there are multiple objects within sensor range of a short wall. In that case, the distance reading for the objects can be used to determine which detections on the long wall refer to which object.

**Heuristic 3:** If  $n$  objects are detected from a short wall, then the first  $n$  detected objects along the next wall refer to those same objects. The order of detection on the long wall corresponds to the distance readings on the short wall in order of increasing distance.

Finally, once two walls have been fully traversed, all objects will have been detected at least once. This suggests a fourth, and perhaps final, heuristic to improve object identification.

**Heuristic 4:** The same objects detected when following the first long wall will appear in reverse order when following the second long wall.

With these heuristics in place, the robot should be able to identify all objects in its environment, offering the potential for full landmark navigation.

One final addition could be made to the landmark mapping scheme. A simple addition to the landmark data would allow for the conversion of the relation-based landmark map to a rough distance-based coordinate map of the environment. If the robot were to record the time between landmarks, these times could be converted to rough distances since the robot tends to move in a smooth, regular fashion.

## Appendix A

Information were downloaded from

<http://www.cs.trinity.edu/ftp/pub/palm/docs/IncsFileChanges.html>

### Palm OS 3.5 Support File Locations

This table details what's new or changed in the SDK file structure since the Palm OS 3.1/Palm VII release. The Notes column shows which files are new, renamed, moved, or deleted.

Palm OS 3.1 Support	PalmVII Support	Palm OS 3.5 Support	Notes
Incs:AppBuildRules.h	Incs:AppBuildRules.h		Moved into BuildDefaults.h and BuildDefines.h
Incs:BuildRules.h	Incs:BuildRules.h		Moved into BuildDefaults.h and BuildDefines.h
		Incs:BuildDefaults.h	Replaces AppBuildRules.h and BuildRules.h
		Incs:BuildDefines.h	Replaces AppBuildRules.h and BuildRules.h
<b>Incs:Common.h</b>	<b>Incs:Common.h</b>		<b>Is now PalmTypes.h (Incs:PalmTypes.h)</b>
Incs:Es.Prefix.h			Moved to (Incs:Libraries:Simulator:Locale:Locale_es ES.h)
Incs:Fr.Prefix.h			Moved to (Incs:Libraries:Simulator:Locale:Locale_frFR.h)
Incs:Ge.Prefix.h			Moved to (Incs:Libraries:Simulator:Locale:Locale_deDE.h)
Incs:Jp.Prefix.h			Moved to (Incs:Libraries:Simulator:Locale:Locale_jpJP.h)
Incs:Pilot.h	Incs:Pilot.h		Now is PalmOS.h (Incs:PalmOS.h)
		<b>Incs:PalmCompatibility.h</b>	<b>New file in 3.5</b>
		Incs:PalmOS.h	Replaces Pilot.h
		Incs:PalmOS.pch	Replaces old Precompiled headers
		<b>Incs:PalmTypes.h</b>	<b>Replaces Common.h (Incs:Common.h)</b>
Incs:SysBuildRules.h	Incs:SysBuildRules.h		Moved to BuildDefines.h (Incs:BuildDefines.h)
Incs:Us.Prefix.h			Moved to (Incs:Libraries:Simulator:Locale:Locale_en)

			US.h)
		Incs:Core:CoreTraps.h	Renamed from SysTraps.h (Incs:System:SysTraps.h)
		<i>Incs:Core:Hardware:HAL.h</i>	<i>New file in 3.5</i>
Incs:Hardware:Hardware.h	Incs:Hardware:Hardware.h		Removed from SDK
		<i>Incs:Core:Hardware:HwrMiscFlags.h</i>	<i>New file in 3.5</i>
Incs:Hardware:M68KHwr.h	Incs:Hardware:M68KHwr.h	Incs:Core:Hardware:M68KHwr.h	
Incs:International:CharLatin.h			Moved to (Incs:Core:System:CharLatin.h)
Incs:International:CharShiftJIS.h			Moved to (Incs:Core:System:CharShiftJIS.h)
Incs:International:FntGlue.h			Moved to (Incs:Libraries:PalmOSGlue:FntGlue.h)
Incs:International:IntlGlue.h			Moved to (Incs:Libraries:PalmOSGlue:IntlGlue.h)
Incs:International:IntlMgr.h	Incs:International:IntlMgr.h		Moved to (Incs:Core:System:IntlMgr.h)
Incs:International:JapaneseFEP.h			
Incs:International:TextMgr.h	Incs:International:TextMgr.h		Moved to (Incs:Core:System:TextMgr.h)
Incs:International:TextServices.h			Renamed to TextServicesMgr.h (Incs:Core:System:TextServicesMgr.h)
Incs:International:TxtGlue.h			Moved to (Incs:Libraries:PalmOSGlue:TxtGlue.h)
Incs:International:WinGlue.h			Moved to (Incs:Libraries:PalmOSGlue:WinGlue.h)
Incs:Precompiled:PalmOSHeaders.mcp	Incs:Precompiled:PalmOSHeaders.mcp		Removed from SDK (See Project Stationery: <b>Palm OS 3.5 Palm OS C App:ReadMe-Precompiled Headers</b> for info)
Incs:Precompiled:PalmSimHeaders.mcp	Incs:Precompiled:PalmSimHeaders.mcp		Removed from SDK
Incs:Precompiled:Pilot.pch	Incs:Precompiled:Pilot.pch		Removed from SDK
Incs:Precompiled:Pilot.pch++	Incs:Precompiled:Pilot.pch++		Removed from SDK
Incs:Precompiled:SysAll.pch	Incs:Precompiled:SysAll.pch		Removed from SDK
Incs:Precompiled:SysAll.pch++	Incs:Precompiled:SysAll.pch++		Removed from SDK
Incs:Precompiled:Obj:Pilot.h++.mch	Incs:Precompiled:Obj:Pilot.h++.mch		Removed from SDK
Incs:Precompiled:Obj:Pilot.h++.sim.mch	Incs:Precompiled:Obj:Pilot.h++.sim.mch		Removed from SDK
Incs:Precompiled:Obj:Pilot.h.mch	Incs:Precompiled:Obj:Pilot.h.mch		Removed from SDK
Incs:Precompiled:Obj:Pilot.h.sim.mch	Incs:Precompiled:Obj:Pilot.h.sim.mch		Removed from SDK
Incs:Precompiled:Obj:SysAll.h++.mch	Incs:Precompiled:Obj:SysAll.h++.mch		Removed from SDK
Incs:Precompiled:Obj:SysAll.h++.sim.mch	Incs:Precompiled:Obj:SysAll.h++.sim.mch		Removed from SDK
<b>Incs:Precompiled:Obj:SysAll.h.mch</b>	<b>Incs:Precompiled:Obj:SysAll.h.mch</b>		<b>Removed from SDK</b>
<b>Incs:Precompiled:Obj:SysAll.h.sim.mch</b>	<b>Incs:Precompiled:Obj:SysAll.h.sim.mch</b>		<b>Removed from SDK</b>
Incs:Precompiled:Obj (English):Pilot.h++.mch			Removed from SDK
Incs:Precompiled:Obj (English):Pilot.h++.sim.mch			Removed from SDK
Incs:Precompiled:Obj (English):Pilot.h.mch			Removed from SDK
Incs:Precompiled:Obj (English):Pilot.h.sim.mch			Removed from SDK
Incs:Precompiled:Obj (English):SysAll.h++.mch			Removed from SDK
Incs:Precompiled:Obj (English):SysAll.h++.sim.mch			Removed from SDK
<b>Incs:Precompiled:Obj (English):SysAll.h.mch</b>			<b>Removed from SDK</b>
<b>Incs:Precompiled:Obj</b>			<b>Removed from SDK</b>

(English):SysAll.h.sim.mch			
Incs:Precompiled:Obj (Japan):Pilot.h++.mch			Removed from SDK
Incs:Precompiled:Obj (Japan):Pilot.h++.sim.mch			Removed from SDK
Incs:Precompiled:Obj (Japan):Pilot.h.mch			Removed from SDK
Incs:Precompiled:Obj (Japan):Pilot.h.sim.mch			Removed from SDK
Incs:Precompiled:Obj (Japan):SysAll.h++.mch			Removed from SDK
Incs:Precompiled:Obj (Japan):SysAll.h++.sim.mch			Removed from SDK
Incs:Precompiled:Obj (Japan):SysAll.h.mch			Removed from SDK
Incs:Precompiled:Obj (Japan):SysAll.h.sim.mch			Removed from SDK
Incs:System:AlarmMgr.h	Incs:System:AlarmMgr.h	Incs:Core:System:AlarmMgr.h	
Incs:System:AppLaunchCmd.h	Incs:System:AppLaunchCmd.h	Incs:Core:System:AppLaunchCmd.h	
		Incs:Core:System:Bitmap.h	New file in 3.5
		Incs:Core:System:CharAttr.h	Moved from (Incs:UI:CharAttr.h)
		Incs:Core:System:CharLatin.h	Moved from (Incs:International:CharLatin.h)
		Incs:Core:System:Chars.h	Moved from (Incs:UI:CharAttr.h)
		Incs:Core:System:CharShifJIS.h	Moved from (Incs:International:CharShifJIS.h)
Incs:System:CMClient.h	Incs:System:CMClient.h		Removed from SDK
Incs:System:CMCommon.h	Incs:System:CMCommon.h	Incs:Core:System:CMCommon.h	
		Incs:Core:System:CMLConst.h	New file in 3.5
		Incs:Core:System:ConnectionMgr.h	New file in 3.5
Incs:System:ConsoleMgr.h	Incs:System:ConsoleMgr.h	Incs:Core:System:ConsoleMgr.h	
Incs:System:Crc.h	Incs:System:Crc.h	Incs:Core:System:Crc.h	
	Incs:System:CTP.h	Incs:Core:System:CTP.h	
Incs:System:DataMgr.h	Incs:System:DataMgr.h	Incs:Core:System:DataMgr.h	
Incs:System:DataPrv.h	Incs:System:DataPrv.h		Removed from SDK
Incs:System:DateTime.h	Incs:System:DateTime.h	Incs:Core:System:DateTime.h	
Incs:System:DebugMgr.h	Incs:System:DebugMgr.h	Incs:Core:System:DebugMgr.h	
Incs:System:DebugPrv.h	Incs:System:DebugPrv.h		Removed from SDK
Incs:System:DLCommon.h	Incs:System:DLCommon.h	Incs:Core:System:DLCommon.h	
Incs:System:DLServer.h	Incs:System:DLServer.h	Incs:Core:System:DLServer.h	
Incs:System:Encrypt.h	Incs:System:Encrypt.h	Incs:Core:System:Encrypt.h	
		Incs:Core:System:ErrorBase.h	New file in 3.5
Incs:System:ErrorMgr.h	Incs:System:ErrorMgr.h	Incs:Core:System:ErrorMgr.h	
Incs:System:ExgLib.h	Incs:System:ExgLib.h	Incs:System:ExgLib.h	
Incs:System:ExgMgr.h	Incs:System:ExgMgr.h	Incs:Core:System:ExgMgr.h	
Incs:System:FatalAlert.h	Incs:System:FatalAlert.h		Moved to (Incs:Core:UI:FatalAlert.h)
Incs:System:FeatureMgr.h	Incs:System:FeatureMgr.h	Incs:Core:System:FeatureMgr.h	
Incs:System:FileStream.h	Incs:System:FileStream.h	Incs:Core:System:FileStream.h	
Incs:System:FloatMgr.h	Incs:System:FloatMgr.h	Incs:Core:System:FloatMgr.h	* Note: The FloatMgr.h in prior SDK's was an old FloatMgr.h. This FloatMgr.h is a new file and contacts no references to the old FloatMgr.h.
Incs:System:Globals.h	Incs:System:Globals.h		Removed from SDK
		Incs:Core:System:Font.h	Moved from (Incs:UI:Font.h)
Incs:System:Graffiti.h	Incs:System:Graffiti.h	Incs:Core:System:Graffiti.h	

Incs:System:GraffitiReference.h	Incs:System:GraffitiReference.h		Moved (Incs:Core:UI:GraffitiReference.h) to
Incs:System:HostControl.h	Incs:System:HostControl.h	Incs:Core:System:HostControl.h	
Incs:System:ImcUtils.h	Incs:System:ImcUtils.h	Incs:Core:System:ImcUtils.h	
	Incs:System:INetMgr.h		Moved to (Incs:Libraries:INet:INetMgr.h)
		Incs:Core:System:IntlMgr.h	Moved from (Incs:International:IntlMgr.h)
Incs:System:irlib.h	Incs:System:irlib.h	Incs:Core:System:IrLib.h	
Incs:System:Keyboard.h	Incs:System:Keyboard.h		Moved to (Incs:Core:UI:Keyboard.h)
Incs:System:KeyMgr.h	Incs:System:KeyMgr.h	Incs:Core:System:KeyMgr.h	
Incs:System:Launcher.h	Incs:System:Launcher.h		Moved to (Incs:Core:UI:Launcher.h)
Incs:System:Localize.h	Incs:System:Localize.h	Incs:Core:System:Localize.h	
Incs:System:MemoryMgr.h	Incs:System:MemoryMgr.h	Incs:Core:System:MemoryMgr.h	
Incs:System:MemoryPrv.h	Incs:System:MemoryPrv.h		Removed from SDK
Incs:System:MemoryPrvNew.h	Incs:System:MemoryPrvNew.h		Removed from SDK
Incs:System:ModemMgr.h	Incs:System:ModemMgr.h	Incs:Core:System:ModemMgr.h	
		<i>Incs:Core:System:NetBitUtils.h</i>	<i>New file in 3.5</i>
Incs:System:NetMgr.h	Incs:System:NetMgr.h	Incs:Core:System:NetMgr.h	
Incs:System:NewFloatMgr.h	Incs:System:NewFloatMgr.h		Moved to FloatMgr.h (Incs:Core:System:FloatMgr.h)
		<i>Incs:Core:System:NotifyMgr.h</i>	<i>New file in 3.5</i>
Incs:System:PalmStdio.h	Incs:System:PalmStdio.h		
		<i>Incs:Core:System:OverlayMgr.h</i>	<i>New file in 3.5</i>
Incs:System>Password.h	Incs:System>Password.h	Incs:Core:System>Password.h	
Incs:System:PenMgr.h	Incs:System:PenMgr.h	Incs:Core:System:PenMgr.h	
Incs:System:Preferences.h	Incs:System:Preferences.h	Incs:Core:System:Preferences.h	
		Incs:Core:System:Rect.h	Moved from (Incs:UI:Rect.h)
		<i>Incs:Core:System:ScriptPlugin.h</i>	<i>New file in 3.5</i>
		<i>Incs:Core:System:SerialDrvr.h</i>	<i>New file in 3.5</i>
Incs:System:SerialLinkMgr.h	Incs:System:SerialLinkMgr.h	Incs:Core:System:SerialLinkMgr.h	
Incs:System:SerialLinkPrv.h	Incs:System:SerialLinkPrv.h		Removed from SDK
Incs:System:SerialMgr.h	Incs:System:SerialMgr.h	Incs:Core:System:SerialMgr.h	
		<i>Incs:Core:System:SerialMgrOld.h</i>	<i>New file in 3.5</i>
		<i>Incs:Core:System:SerialSdrv.h</i>	<i>New file in 3.5</i>
		<i>Incs:Core:System:SerialVdrv.h</i>	<i>New file in 3.5</i>
Incs:System:SoundMgr.h	Incs:System:SoundMgr.h	Incs:Core:System:SoundMgr.h	
		Incs:Core:System:StdIOPalm.h	
		Incs:Core:System:StdIOProvider.h	
	Incs:System:SoundPrv.h		Removed from SDK
Incs:System:StringMgr.h	Incs:System:StringMgr.h	Incs:Core:System:StringMgr.h	
Incs:System:SysAll.h	Incs:System:SysAll.h		
		Incs:Core:System:SysEvent.h	Moved from (Incs:UI:Event.h)
Incs:System:SysEvtMgr.h	Incs:System:SysEvtMgr.h	Incs:Core:System:SysEvtMgr.h	
		Incs:Core:System:SysResTypes.rh	
Incs:System:SystemMgr.h	Incs:System:SystemMgr.h	Incs:Core:System:SystemMgr.h	
Incs:System:SystemMgr.rh	Incs:System:SystemMgr.rh		
Incs:System:SystemPkt.h	Incs:System:SystemPkt.h	Incs:Core:System:SystemPkt.h	
Incs:System:SystemPrv.h	Incs:System:SystemPrv.h		Removed from SDK
		Incs:Core:System:SystemPublic.h	
		Incs:Core:System:SystemResources.h	

Incs:System:SysTraps.h	Incs:System:SysTraps.h		Renamed (Incs:Core:CoreTraps.h) CoreTraps.h
Incs:System:SysUtils.h	Incs:System:SysUtils.h	Incs:Core:System:SysUtils.h	
		Incs:Core:System:TextMgr.h	Moved from (Incs:International:TextMgr.h)
		Incs:Core:System:TextServicesMgr.h	Was (Incs:International:TextServices.h)
Incs:System:TimeMgr.h	Incs:System:TimeMgr.h	Incs:Core:System:TimeMgr.h	
		Incs:Core:System:Window.h	Moved from (Incs:UI:Window.h)
Incs:System:Unix:arpa_inet.h	Incs:System:Unix:arpa_inet.h	Incs:Core:System:Unix:arpa_inet.h	
Incs:System:Unix:netdb.h	Incs:System:Unix:netdb.h	Incs:Core:System:Unix:netdb.h	
Incs:System:Unix:netinet_in.h	Incs:System:Unix:netinet_in.h	Incs:Core:System:Unix:netinet_in.h	
Incs:System:Unix:netinet_in_sysm.h	Incs:System:Unix:netinet_in_sysm.h	Incs:Core:System:Unix:netinet_in_sysm.h	
Incs:System:Unix:netinet_ip.h	Incs:System:Unix:netinet_ip.h	Incs:Core:System:Unix:netinet_ip.h	
Incs:System:Unix:netinet_tcp.h	Incs:System:Unix:netinet_tcp.h	Incs:Core:System:Unix:netinet_tcp.h	
Incs:System:Unix:sys_errno.h	Incs:System:Unix:sys_errno.h	Incs:Core:System:Unix:sys_errno.h	
Incs:System:Unix:sys_socket.h	Incs:System:Unix:sys_socket.h	Incs:Core:System:Unix:sys_socket.h	
Incs:System:Unix:sys_socketvar.h	Incs:System:Unix:sys_socketvar.h	Incs:Core:System:Unix:sys_socketvar.h	
Incs:System:Unix:sys_time.h	Incs:System:Unix:sys_time.h	Incs:Core:System:Unix:sys_time.h	
Incs:System:Unix:sys_types.h	Incs:System:Unix:sys_types.h	Incs:Core:System:Unix:sys_types.h	
Incs:System:Unix:sys_uio.h	Incs:System:Unix:sys_uio.h	Incs:Core:System:Unix:sys_uio.h	
Incs:System:Unix:unix_fcntl.h	Incs:System:Unix:unix_fcntl.h	Incs:Core:System:Unix:unix_fcntl.h	
Incs:System:Unix:unix_netdb.h	Incs:System:Unix:unix_netdb.h	Incs:Core:System:Unix:unix_netdb.h	
Incs:System:Unix:unix_stdarg.h	Incs:System:Unix:unix_stdarg.h	Incs:Core:System:Unix:unix_stdarg.h	
Incs:System:Unix:unix_stdio.h	Incs:System:Unix:unix_stdio.h	Incs:Core:System:Unix:unix_stdio.h	
Incs:System:Unix:unix_stdlib.h	Incs:System:Unix:unix_stdlib.h	Incs:Core:System:Unix:unix_stdlib.h	
Incs:System:Unix:unix_string.h	Incs:System:Unix:unix_string.h	Incs:Core:System:Unix:unix_string.h	
Incs:UI:AboutBox.h	Incs:UI:AboutBox.h	Incs:Core:UI:AboutBox.h	
Incs:UI:Category.h	Incs:UI:Category.h	Incs:Core:UI:Category.h	
Incs:UI:CharAttr.h	Incs:UI:CharAttr.h		Moved to (Incs:Core:System:CharAttr.h)
Incs:UI:Chars.h	Incs:UI:Chars.h		Moved to (Incs:Core:System:Chars.h)
Incs:UI:Clipboard.h	Incs:UI:Clipboard.h	Incs:Core:UI:Clipboard.h	
Incs:UI:Contrast.h	Incs:UI:Contrast.h		Moved to (Incs:Core:UI:UIControls.h)
Incs:UI:Control.h	Incs:UI:Control.h	Incs:Core:UI:Control.h	
Incs:UI:Day.h	Incs:UI:Day.h	Incs:Core:UI:Day.h	
Incs:UI:Event.h	Incs:UI:Event.h	Incs:Core:UI:Event.h	
		Incs:Core:UI:FatalAlert.h	Moved from (Incs:System:FatalAlert.h)
Incs:UI:Field.h	Incs:UI:Field.h	Incs:Core:UI:Field.h	
Incs:UI:Find.h	Incs:UI:Find.h	Incs:Core:UI:Find.h	
Incs:UI:Font.h	Incs:UI:Font.h		Moved to (Incs:Core:System:Font.h)
Incs:UI:FontSelect.h	Incs:UI:FontSelect.h	Incs:Core:UI:FontSelect.h	
Incs:UI:Form.h	Incs:UI:Form.h	Incs:Core:UI:Form.h	
		Incs:Core:UI:GraffitiReference.h	Moved from (Incs:System:GraffitiReference.h)
Incs:UI:GraffitiShift.h	Incs:UI:GraffitiShift.h	Incs:Core:UI:GraffitiShift.h	
	Incs:UI:GraffitiUI.h		
Incs:UI:Init.h	Incs:UI:Init.h		Removed from SDK
Incs:UI:InsPoint.h	Incs:UI:InsPoint.h	Incs:Core:UI:InsPoint.h	
		Incs:Core:UI:Keyboard.h	Moved from (Incs:System:Keyboard.h)
		Incs:Core:UI:Launcher.h	Moved from (Incs:System:Launcher.h)
Incs:UI:List.h	Incs:UI:List.h	Incs:Core:UI:List.h	

Incs:UI:Menu.h	Incs:UI:Menu.h	Incs:Core:UI:Menu.h	
Incs:UI:PhoneLookup.h	Incs:UI:PhoneLookup.h	Incs:Core:UI:PhoneLookup.h	
		<i>Incs:Core:UI:PrivateRecords.h</i>	<i>New file in 3.5</i>
Incs:UI:Progress.h	Incs:UI:Progress.h	Incs:Core:UI:Progress.h	
Incs:UI:Rect.h	Incs:UI:Rect.h		Moved to (Incs:Core:System:Rect.h)
Incs:UI:ScrDriver.h	Incs:UI:ScrDriver.h		Removed from SDK

## Appendix B

Here are some tips we found to help update data types to Palm OS 3.5:

- 1) Compile the file.
- 2) If you get an error, which indicate an undefined or invalid data type, then go to the file
- 3) Right click on the highlighted key word, which gives you an error.
- 4) Then click on “Go to typedef decleration of [the name of the key word]”
- 5) This takes you to the header file, where it is defined. If the header file is *Pilot.h*, *Common.h*, or *SysAll.h*, you need to find the new equivalent using the conversion chart (Appendix A).
- 6) Then type in the new keyword, Right click on it, and select “go to type definition”
- 7) This takes you to where the type is defined and you can get more precise information.

The major type definition you have to change are primitive types (char, short, long).

More complete information about conversion of the data types can be found on pages 88-89 of the Palm OS Bible.