Damian Nowak
Łukasz Dadlez
(zespół nr 6)

25.11.2011r.

Laboratorium Inteligentnych Maszyn i Systemów
Wydział Elektryczny Politechniki Warszawskiej

Sterowanie zaawansowanym autonomicznym robotem mobilnym (Pioneer)

Sprawozdanie

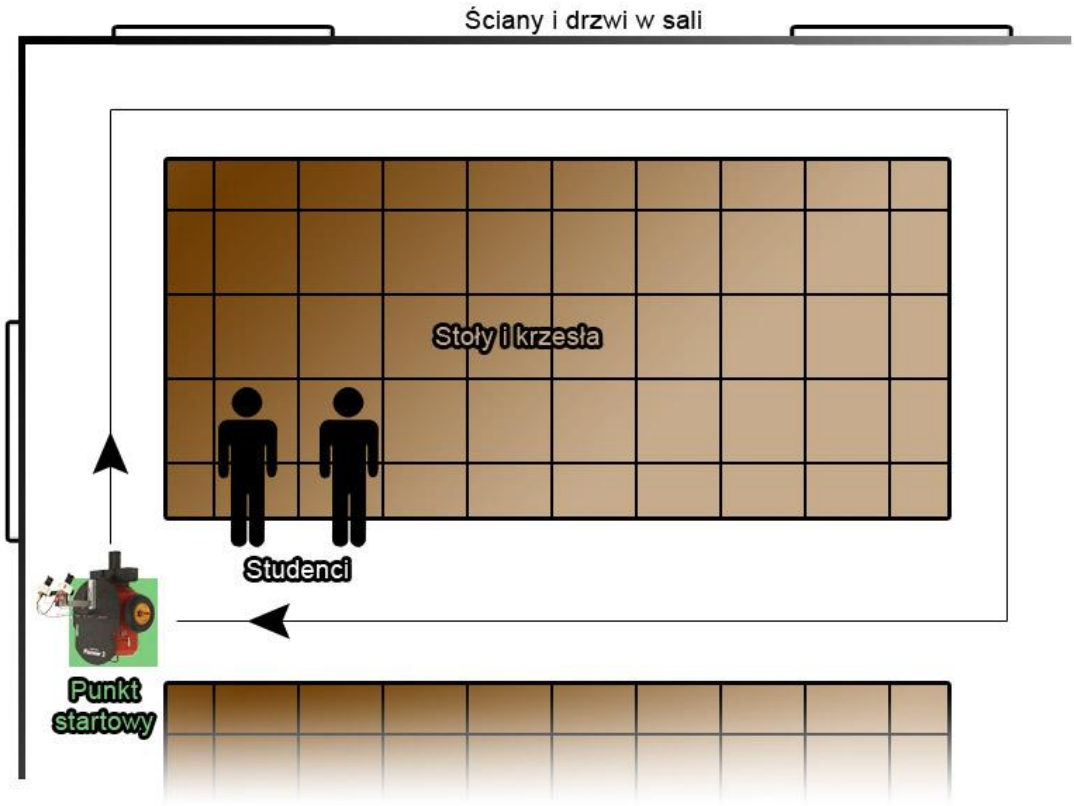
Przebieg ćwiczenia

Część pierwsza

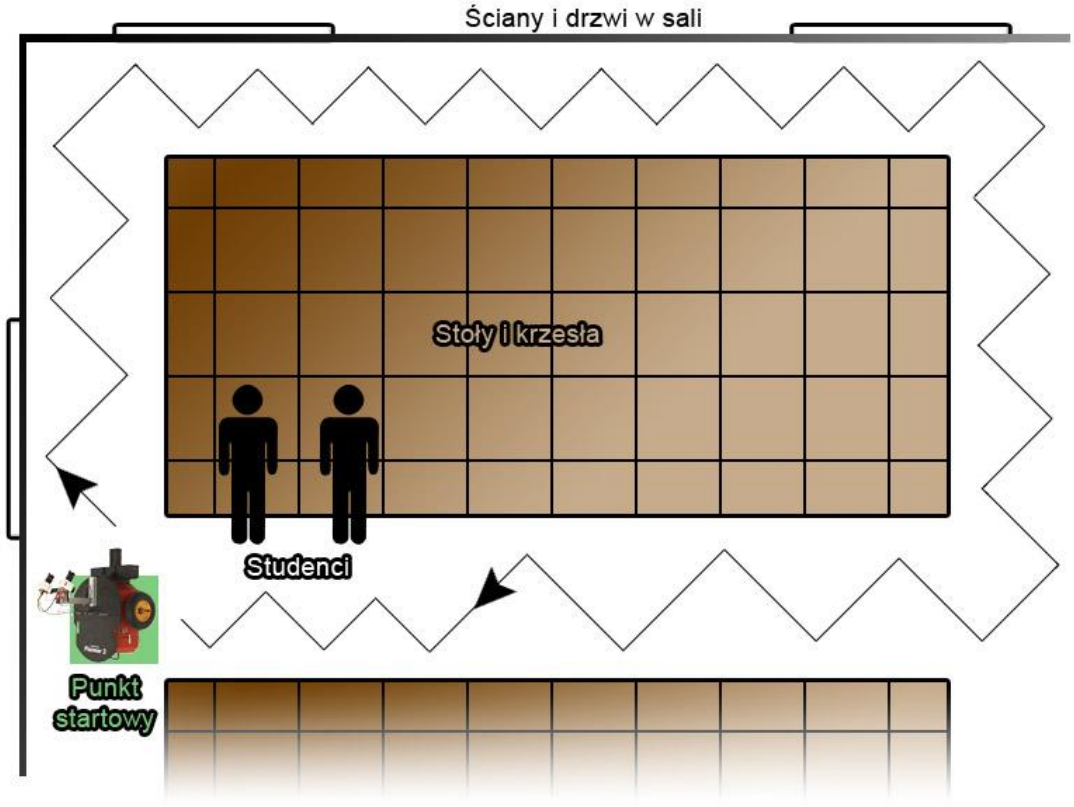
Podstawowym zadaniem w pierwszej części ćwiczenia było zapoznanie się z obsługą i właściwościami jezdnymi robota, jego zdolnością do wykrywania przeszkód oraz dokładnością nawigacji odometrycznej.

Robota wyposażono w 16 sonarów zamontowanych dookoła niego, z czego do testów zaprogramowano obsługę tylko przedniej części. Testowano reakcję robota na odczyty z sonarów dla różnych przeszkód – ścian, narożników, plecaków, nóg od krzeseł, prostych i pochyłych nóg od stołu, własnych kończyn itp. Dla odległości większej od 20cm, która jest minimalnym zasięgiem czujników, reagował on prawidłowo na każdą przeszkodę, problem sprawiały mu tylko przeszkody pochyłe w postaci przekrzywionych plecaków, niektórych nóg od stołu oraz poprzeczki stołów i krzeseł umieszczone na wysokości mniejszej niż sonary. Reakcja na dodatkowe czujniki w postaci przełączników w zderzakach była nieprawidłowa – pomimo najechania zderzakiem na przeszkodę, robot cały czas na nią napierał, czasami robił to skokowo.

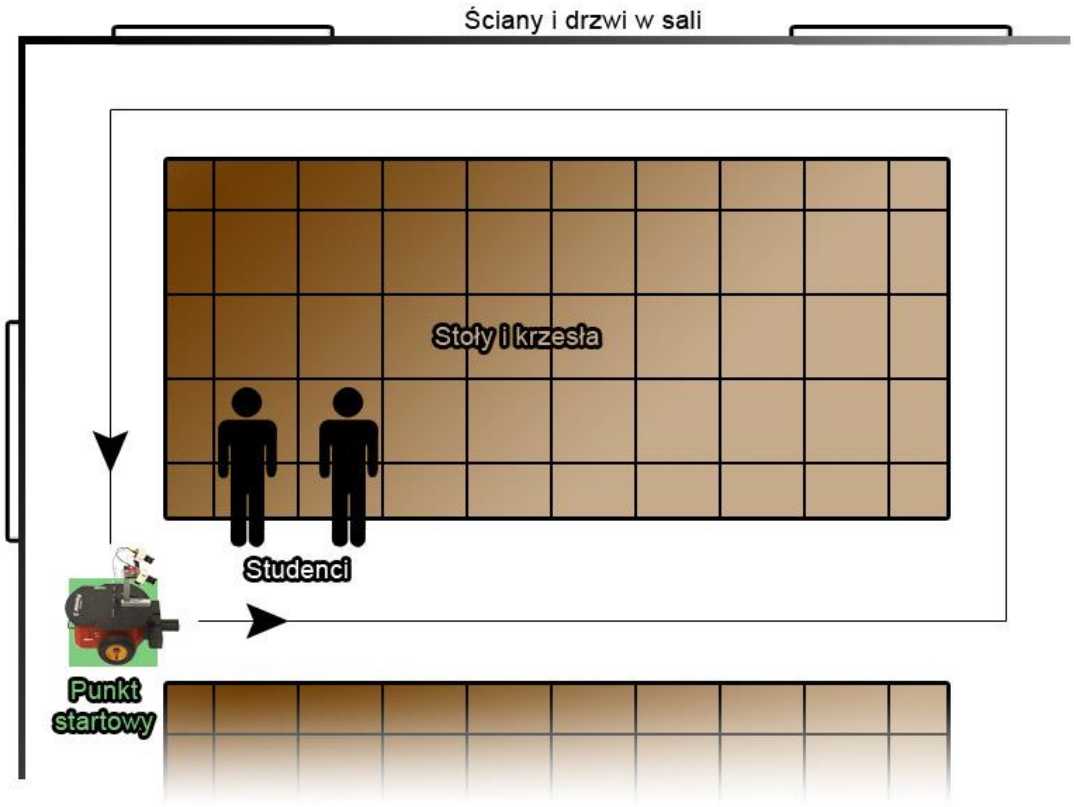
W celu zbadania dokładności nawigacji odometrycznej, wykorzystano zmodyfikowany tzw. test kwadratu wykonany dla kierunku jazdy zgodnego oraz przeciwnego do kierunku ruchu wskazówek zegara. Ustawiono robota w miejscu oznaczonym na ziemi, wyzerowano w programie położenie robota ($X_0 = 0$, $Y_0 = 0$, $\Theta_0 = 0$) i przejechano nim dookoła sali, wracając do punktu wyjścia.



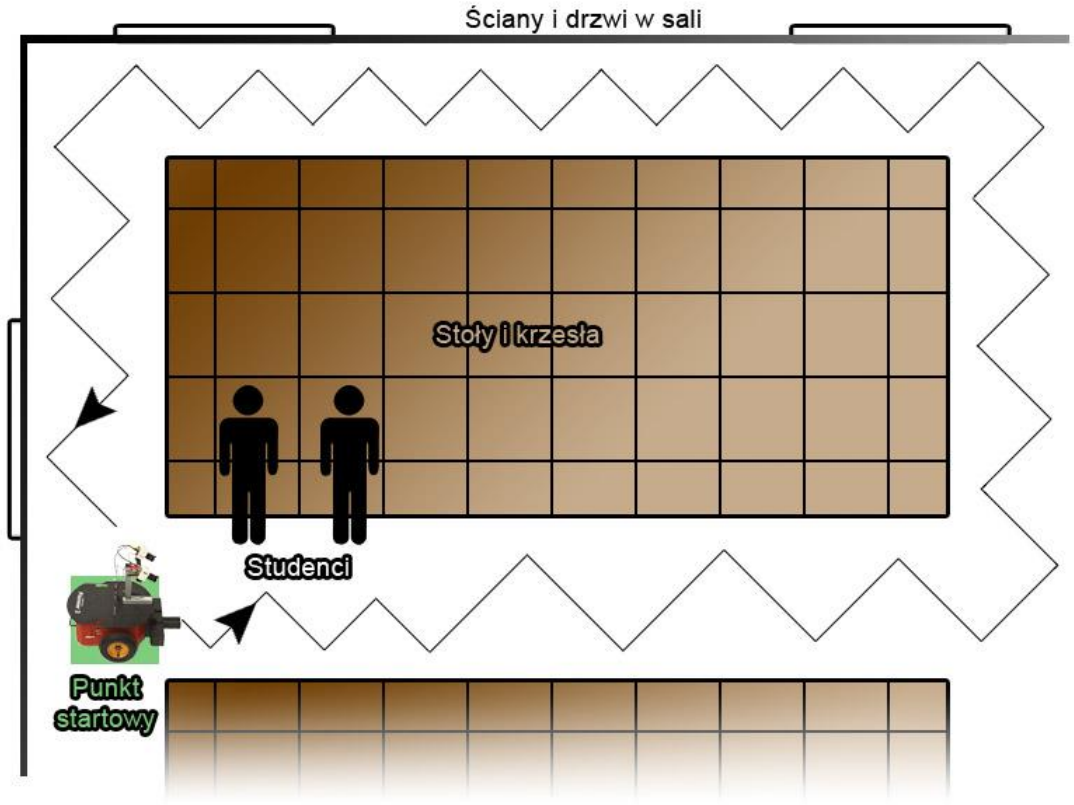
Jazda po linii prostej, zgodnie z ruchem wskazówek zegara



Jazda slalomem, zgodnie z ruchem wskazówek zegara



Jazda po linii prostej, w kierunku przeciwnym do ruchu wskazówek zegara



Jazda slalomem, w kierunku przeciwnym do ruchu wskazówek zegara

Zanotowano następujące wartości:

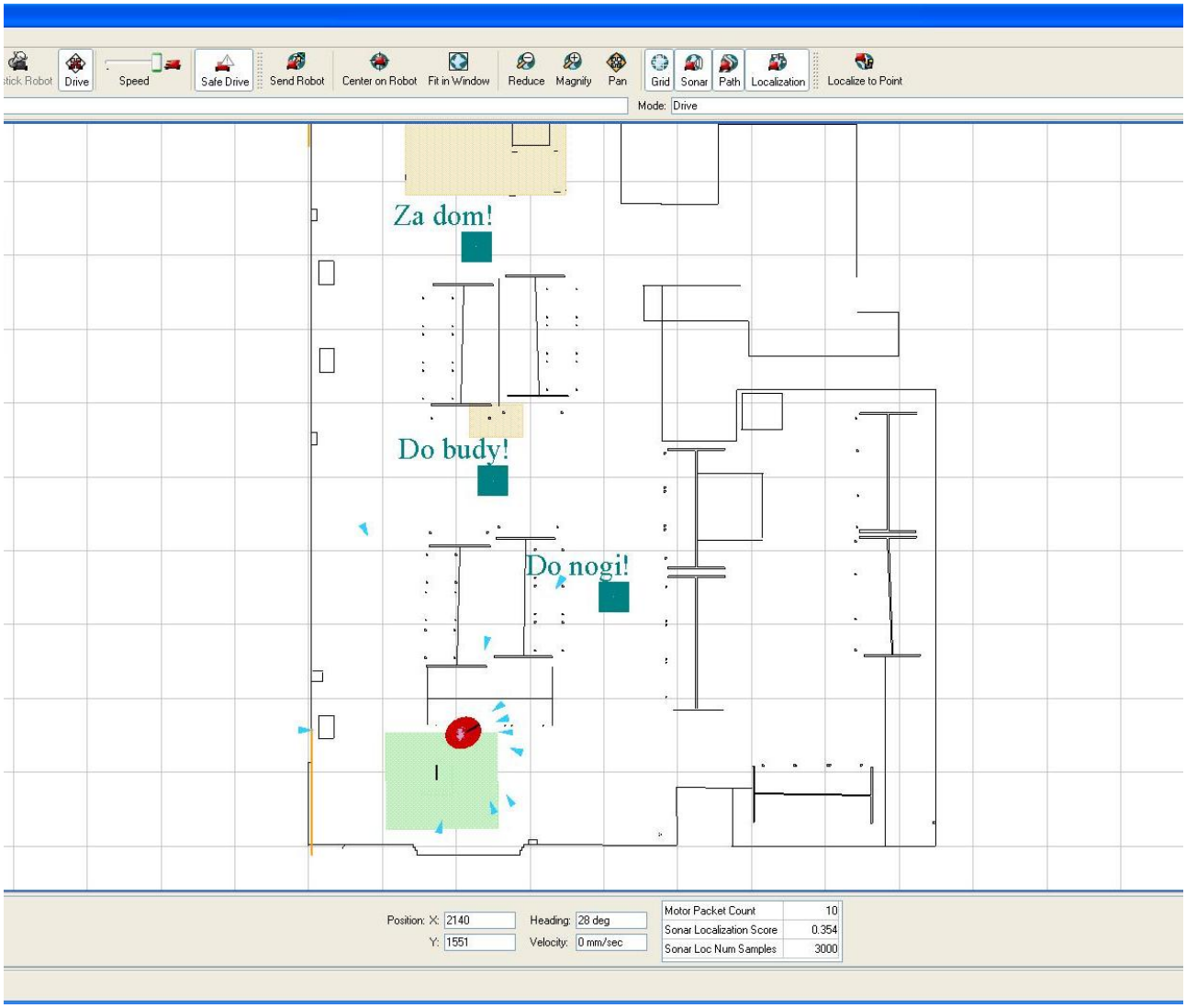
Kierunek ruchu (widok z góry)	**Rodzaj jazdy**	**Wartości sczytane po powrocie robota do punktu początkowego**
Zgodnie z ruchem wskazówek zegara	Po linii prostej	$X_K = 567$ $Y_K = 220$ $\Theta_K = -10,5$
	Slalomem	$X_K = 1809$ $Y_K = 650$ $\Theta_K = -29,4$
Przeciwnie do ruchu wskazówek zegara	Po linii prostej	$X_K = 68$ $Y_K = 81$ $\Theta_K = 4$
	Slalomem	$X_K = 1$ $Y_K = 103$ $\Theta_K = 2,5$

Część druga

Zadaniem drugiej części ćwiczenia było testowanie nawigacji robota z wykorzystaniem odczytu sygnałów z sonarów i mapy otoczenia przy pomocy programu Mobile Eyes.

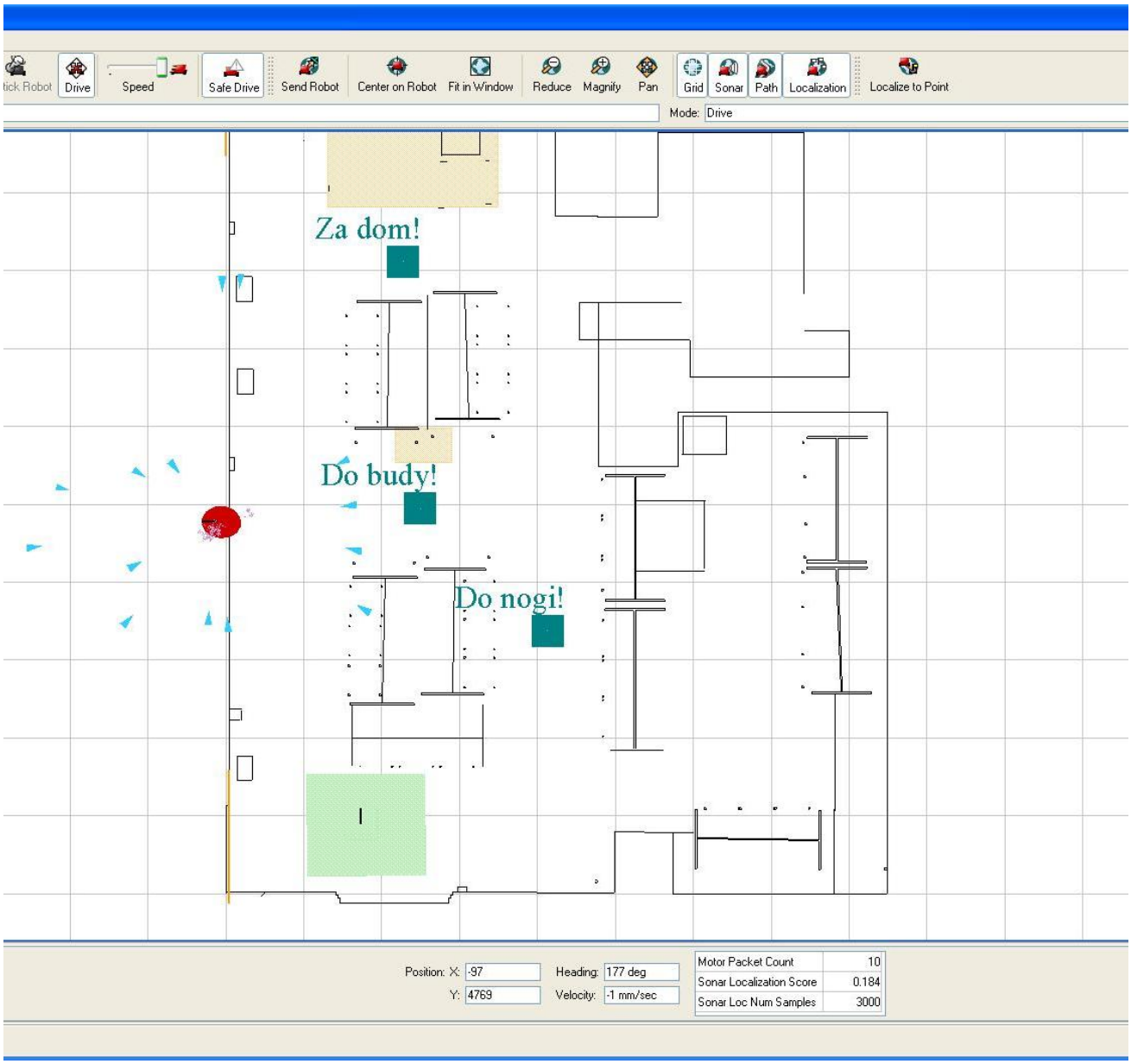
Prób samolokalizacji robota na mapie dokonano w czterech etapach:

1) w programie zaznaczono położenie i orientację początkową możliwie mocno zbliżone do rzeczywistych – robot po kilku ruchach aktualizował swoją pozycję zmniejszając obszar „chmury” do jednego, dwóch lub trzech punktów położonych blisko siebie, ruch po sali był dobrze przekazywany na obraz mapy;



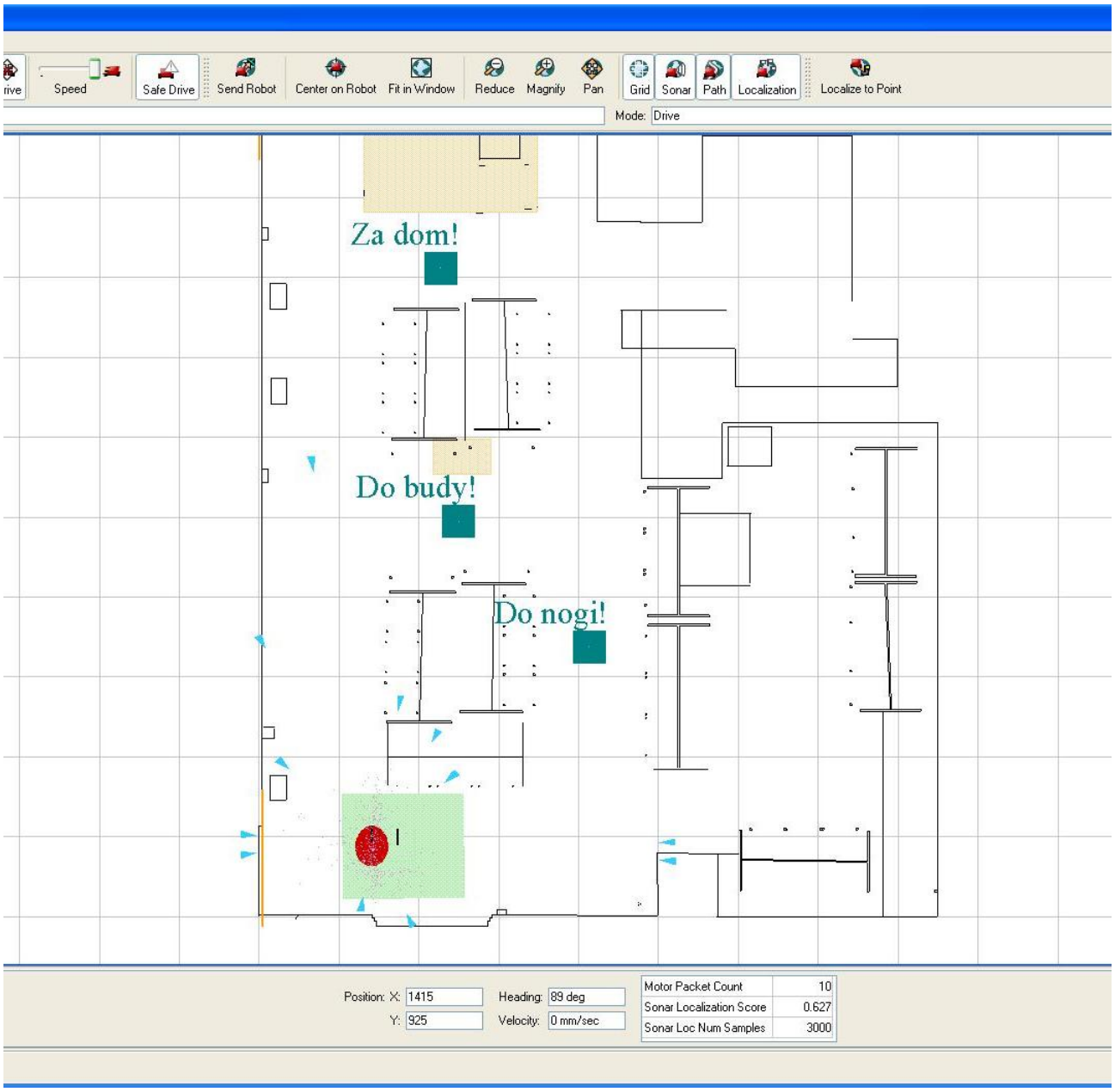
Prawidłowo odczytana lokalizacja robota w zaledwie kilku szybkich ruchach

2) w programie zaznaczono położenie robota możliwie mocno zbliżone do rzeczywistego, natomiast podano orientację zmienioną o 90^o i 180^o względem rzeczywistej – o ile przy kącie 90^o robot po pewnym czasie był w stanie się zlokalizować, to 180^o już nie zawsze;



Błędnie odczytane położenie przy orientacji o 180^o zmienionej względem rzeczywistej

3) w programie zaznaczono orientację zgodną z rzeczywistą, ale zmieniono położenie – dla punktów położonych w obszarze „chmury”, robot bez większych problemów znajdował swoje rzeczywiste położenie, natomiast dla punktów rozmieszczonych na obszarze całej sali raz mu się to udawało, a raz nie;



Widoczna początkowa „chmura”, na obszarze której robot najszybciej się lokalizował

4) przy wprowadzaniu błędnego położenia i orientacji, zachowanie robota było zróżnicowane – zwykle im podawane wartości bardziej odpowiadały rzeczywistemu położeniu i orientacji, tym robot rzadziej się mylił i lokalizował w mniejszej ilości ruchów. Dodatkowo, robot częściej lokalizował się na mapie, gdy znajdował się w korytarzu z lewej strony sali (patrząc od strony wejścia) niż będąc po stronie lewej. Problemy sprawiało mu też położenie rzeczywiste w pobliżu studentów i orientacja przodem do wnętrza narożników sali.

Ostatnia próba dotyczyła obserwacji robota po prawidłowym zlokalizowaniu, poruszającego się na zasadzie wyznaczania punktów docelowych na mapie. Testowano zachowanie robota w przypadku pojawienia się przeszkód nie oznaczonych na mapie (wysuniętych krzeseł, nóg, plecaków itp.). Przy zachowaniu minimalnego prześwitu ok. 5÷10cm z każdej strony robota (łączna szerokość korytarza ok. 50cm), był on w stanie ominąć każdą przeszkodę.

Jazda przez slalom ustawionych prostopadle do stołów krzeseł czasami sprawiała trudności, robot radził sobie przy większych prześwitach między krzesłami:



Natomiast po ustawieniu krzeseł pod kątem 45^{o}, robot był już w stanie przejechać przez slalom niemal bezbłędnie:





Część trzecia

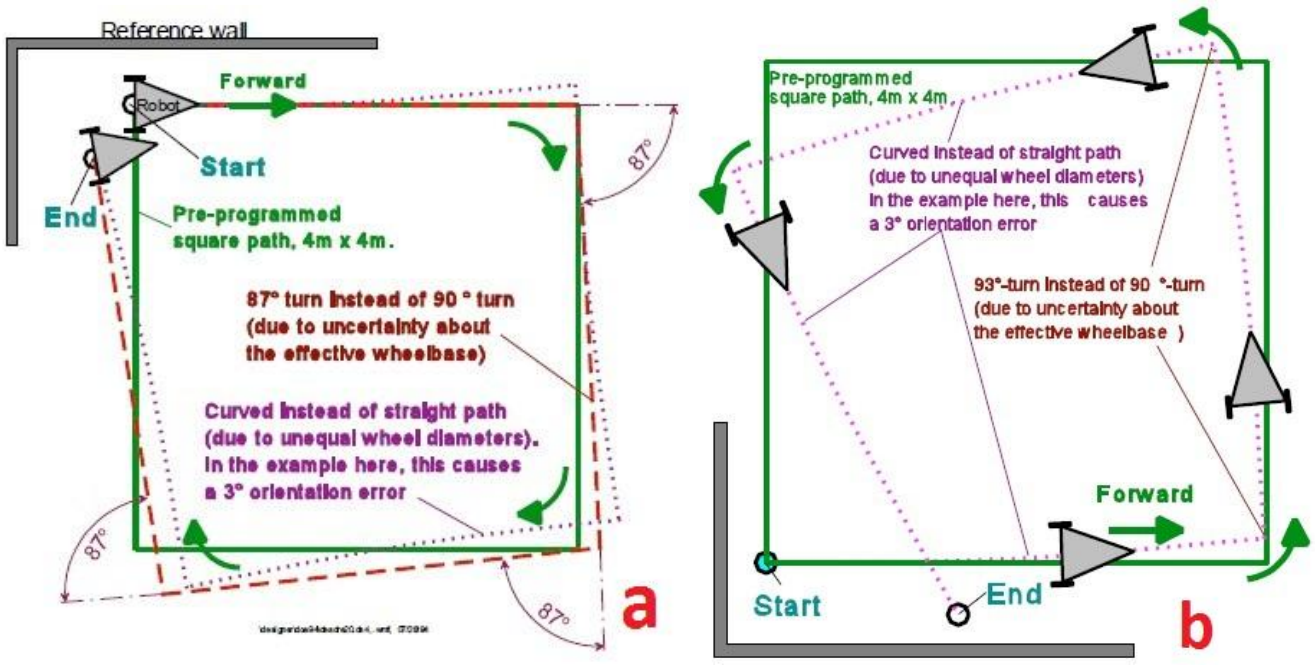
Trzecim zadaniem ćwiczenia było zmodyfikowanie kodu programu odpowiadającego zachowaniu robota. Efektem końcowym miał być program umożliwiający jazdę z omijaniem przeszkód na drodze.

Zrealizowany program spełnił założenia, testy pokazały, że działa prawidłowo.

Wnioski

„Test kwadratu" pokazał, że błędy w dokładności nawigacji odometrycznej są bardzo duże, zwłaszcza dla ruchu w kierunku zgodnym do ruchu wskazówek zegara. Dodatkowo, błędy były zdecydowanie większe podczas jazdy slalomem, niż podczas jazdy po linii prostej.

Przyczyny takich różnic w błędach dla różnych kierunków jazdy, pokazują poniższe rysunki:



a) – ruch zgodny, b) – ruch przeciwny do wskazówek zegara (w naszym przypadku błąd zwiększał się dla ruchu w kierunku zgodnym do ruchu wskazówek zegara, ale geneza błędu jest ta sama)

Źródło: UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots
By Johann Borenstein and Liqiang Feng, The University of Michigan

Łatwo sobie wyobrazić, jak dodatkowo wpływał na taki test ruch slalomem zamiast po linii prostej – każda zmiana kierunku generowała dodatkowy błąd. Należy również dodać, że testowana przez nas trasa nie miała kształtu kwadratu, a skręty nie zawsze odbywały się o 90^{o}, ze względu na różne przeszkody. Inne możliwe przyczyny tak dużych błędów, to:

- różnica między średnicami obu kół,
- dokładność pomiaru rozstawu kół,
- brak zbieżności kół,
- skończona rozdzielczość enkoderów,
- nierówna nawierzchnia,
- drobne przeszkody („uskoki”, pojedyncze drobiny piasku) na drodze,
- poślizg kół przy ruszaniu, obracaniu się, zatrzymywaniu, tarcie szerokich opon.

Problemy samolokalizacji robota w drugiej części ćwiczenia, były przede wszystkim spowodowane wpływem pojawienia się przeszkód nie ujętych na mapie (studenci, poprzesuwane krzesła), niejednoznaczności otrzymywanych danych (np. zbliżone odczyty z sonarów dla dwóch lub więcej miejsc w sali), niedokładności mapy otoczenia oraz krótkiego czasu trwania ćwiczenia – gdyby robot miał możliwość sprawdzenia każdej możliwej pozycji, prawdopodobnie udałoby mu się prawidłowo zlokalizować.

Wpływ ustawienia krzeseł przy przejeździe przez „tunel” miał znaczenie ze względu na umiejętność wykrycia prześwitu między nogami krzesła przez robota. Gdy robot widział na zakrętach tylko jedną nogę (krzesła ustawione pod kątem), reagował prawidłowo.

Dodatki

Listing programu z części trzeciej:

```
#include "Aria.h"
#include "screen.h"
ArRobot *robot;
int nrsumy;
int maks;

void sonarPrinter(void)
{
   clrscr();
   int i;
   int tablica[8];
   int sumy[7];

   gotoxy(0,0);
   for (i = 0; i < 8; i++){
         if (robot->getSonarRange(i) <5000) {
               tablica[i] = robot->getSonarRange(i);
         } else {
               tablica[i] = 0;
         }
   }
```

```
   for (i=0; i<7; i++){
         sumy[i]= tablica[i] + tablica[i+1];
   }
   for (i = 0; i<8;i++){
         printf(" %4d", tablica[i]);
   }
   printf("\n");
   for (i = 0; i<7;i++){
         printf(" %d", sumy[i]);
   }
   printf("\n");
   maks = 0;
   for (i = 0; i<7;i++){
         if (sumy[i] >= maks){
               maks = sumy[i];
               nrsumy = i;
         }
   }
   printf(" %d\n",maks);
   printf(" %d\n",nrsumy);
   printf("\n\n\n\n\n\n\n");
   fflush(stdout);

}
int main(int argc, char **argv)
{
   Aria::init();
   ArArgumentParser parser(&argc, argv);
   ArSimpleConnector simpleConnector(&parser);
   robot = new ArRobot;

   ArKeyHandler keyHandler;
   Aria::setKeyHandler(&keyHandler);
   robot->attachKeyHandler(&keyHandler);   // obsluga klawisza ESC
   ArGlobalFunctor sonarPrinterCB(&sonarPrinter);

   // polaczenie z robotem /connection to the robot
   if (!simpleConnector.connectRobot(robot))
   {
         printf("Could not connect to robot... exiting\n");
         Aria::exit(1);
   }
   robot->runAsync(true);

   //robot->addUserTask("Sonar printer", 50, &sonarPrinterCB);

   while(1){
   // uruchomienie silnikow /starting engines
   robot->enableMotors();

   //robot->move(300); //rusz 300 mm do przodu / move 300 mm forwards
   ////printf("ruszam");
   //printf("moving");
   //while ( !robot->isMoveDone() );

   //robot->setDeltaHeading(90);   //obroc o 90 stopni / turn 90 degrees
   //while(!robot->isHeadingDone());

   sonarPrinter();
```

```
	switch (nrsumy){
		case (0):
			robot->setDeltaHeading(70);   //obroc o 90 stopni / turn 90 degrees
			while(!robot->isHeadingDone());
			break;
		case(1):
			robot->setDeltaHeading(40);   //obroc o 90 stopni / turn 90 degrees
			while(!robot->isHeadingDone());
			break;
		case(2):
			robot->setDeltaHeading(20);   //obroc o 90 stopni / turn 90 degrees
			while(!robot->isHeadingDone());
			break;
		case(3):
			robot->setDeltaHeading(00);   //obroc o 90 stopni / turn 90 degrees
			while(!robot->isHeadingDone());
			break;
		case(4):
			robot->setDeltaHeading(-20);   //obroc o 90 stopni / turn 90 degrees
			while(!robot->isHeadingDone());
			break;
		case(5):
			robot->setDeltaHeading(-40);   //obroc o 90 stopni / turn 90 degrees
			while(!robot->isHeadingDone());
			break;
		case(6):
			robot->setDeltaHeading(-70);   //obroc o 90 stopni / turn 90 degrees
			while(!robot->isHeadingDone());
			break;
	}
	if (maks>1000) {
		robot->move(100); //rusz 300 mm do przodu / move 300 mm forwards
	}
	//printf("ruszam");
	//printf("moving");
	while ( !robot->isMoveDone() );

	}


	robot->unlock();

	// wylaczenie robota /robot shutdown
	Aria::exit();
	return 0;
}
```