รายการอ้างอิง

1. กมล ลิมธัญญกูล. การหาแบบจำลองพลวัตและการควบคุมการเคลื่อนที่แบบ VSC ของแขนกล. โครงงานวิศวกรรมไฟฟ้า: 2540.

2. Ammeraal, L. Computer Graphics for Java Programmenrs. New York: John Willey&Sons, 1998.

3. Abdallah, C., Dawson D., and Jamshidi, M. Survey of Robust Control of Rigid Robots. IEEE Control Systems (February 1991): 24–30.

4. Cook, G.E., Biegl, C., Springfield, J.F., and Fernandez K.R. An Intelligent Robotics Simulation. Ind. App. Society Annual Meeting 1994 3 Conference Record of the 1994 IEEE , (1994): 1793–1800.

5. Corke, Peter I. A Robotics Toolbox for MATLAB. IEEE Robotics&Automation Magazine 3 No. 1 (March 1996): 24–32.

6. Decarlo, R.A., Zak, S.H. and Mathews, G.P. Variable Structure Control of Nonlinear Multivariable Systems: A tutorial. Proceeding of IEEE 76 No. 3 (March 1988): 212–232.

7. Gourdeau, R. Object-Oriented Programming for Robotic Manipulator Simulation. IEEE Robotics & Automation Magazine (September 1997): 21–29.

8. Harlow, E. Developing Linux Application with GTK+ and GDK. United States of America: New Riders Publishing, February 1999.

9. Hung, J.Y., Gao, W. and Hung, J.C. Variable Structure Control: A Survey. IEEE Trans. On Industrial Electronics: 40 No. 1 (February 1993): 2–22.

10. Loffler, M., Costescu N., Zergeroglu E., and Dawson D. QRobot-A Multitasking PC Based Robot Control System. Microcomputer Applications Journal 18 No. 1 (1999): 13–22.

11. Mahefka, D.W. and Orin D.E. Xanimate: An Educational Tool for Robot Graphical Simulation. IEEE Robotics & Automation Magazine (March 1996): 6–14.

12. Muraca, P. and Pugliese P. A Variable-structure Regulator for Robotic Systems. Automatica 33 No. 7 (1997): 1423–1426.

13. Murray, J.J. and Neuman, C.P. Organizing Customized Robot Dynamics Algorithms for Efficient Numerical Evaluation. Systems, Man and Cybernetics, IEEE Transactions on: 18 No. 1, (Jan.-Feb. 1988): 115–125.

รายการอ้างอิง (ต่อ)

14. Nethery, J.F. and Spong M.W. Robotica: A mathematica package for robot Analysis. IEEE Robotics & Automation Magazine 1 (March 1994): 13–20.

15. Ortega, R. and Spong, M.W. Adaptive Motion Control of Rigid Robots: A Tutorial. Conf. on Decision and Control (December 1988): 1575–1584.

16. Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. Numerical Recipes in C: the art of scientific computing. 2nd ed. New York: Cambridge University Press, 1994.

17. Schilling, R.J. Fundamentals of Robotics Analysis And Control. New Jersey: Prentice-Hall, 1990.

18. Sciavicco, L. and Siciliano, B. Modeling and Control of Robot Manipulator. New York: McGraw-Hill, 1996.

19. Shyu, K.K., Chu, P.H. and Shang L.J. Control of rigid robot manipulators via combination of adaptive sliding mode control and compensated inverse dynamics approach. IEE Proc. Control Theory Appl 143 No. 3 (May 1997): 238–288.

20. Spong, M.W., and M. Vidyasagar. Robot Dynamics and Control. New York: Wiley, 1989.

21. Su, C.Y. and Leung, T.P. A Sliding Controller with Bound Estimation for Robot Manipulators. IEEE Trans. on Robotics & Automation 9 No. 2 (April 1993): 208–214.

22. Vukobratovic, M. and Potkonjak V. Dynamics of Manipulation Robots: Theory and Application. New York: Springer-Verlag, Scientific Fundamentals of Robotics, vol. 1, 1982.

23. Vukobratovic, M. and Kireanski, M. Kinematics and Trajectory Synthesis of Manipulation Robots. New York: Springer-Verlag, Scientific Fundamentals of Robotics, vol. 3, 1985.

24. Vukobratovic, M. and Stokic D. Real-Time Dynamics of Manipulation Robots. New York: Springer-Verlag, Scientific Fundamentals of Robotics, vol. 4, 1985.

25. Young, K.D., Utkin, V.I., and Ozguner, U. A Control Engineer's Guide to Sliding Mode Control. Control Systems Technology, IEEE Transactions on 7 No. 3 (May 1999): 328–342.

26. Yu, H. and Lloyd, S. Variable structure control of robot manipulators. IEE Proc. Control Theory Appl. 144 No. 2 (March 1997): 167–176.

27. Yu, H. and Seneviratne, L.D., and Earles, S.W.E. Exponentially stable robust control law for robot manipulators. IEE Proc. Control Theory Appl. 141 No. 6 (November 1994): 389–395.

ภาคผนวก

ภาคผนวก ก

คลังโปรแกรมแขนหุ่นยนต์

ก.1 การติดตั้งและการใช้งานคลังโปรแกรม

1. นำไฟล์ต้นฉบับทั้งหมดมาทำการแปลโดยใช้คำสั่ง

   $ gcc -c -fPIC *.c

2. ทำการเชื่อมโยงไฟล์วัตถุ (object file) ทั้งหมดเข้าด้วยกัน เป็นคลังโปรแกรม

   $ gcc -shared -o librobot.so *.o

3. เพื่อความสะดวกในการใช้งาน ควรทำสำเนาไฟล์ robot.h และ librobot.so เก็บไว้ ในไดเร็คทอรี /usr/include หรือ /usr/local/include และ /usr/lib หรือ /usr/local/lib ตามลำดับ

   # cp robot.h /usr/include

   # cp librobot.so /usr/lib

4. ในการใช้งานคลังโปรแกรมนี้ ผู้ใช้มีหน้าที่เขียนไฟล์ภาษา C สมมติว่าชื่อ demo.c ซึ่งในไฟล์นี้มีฟังก์ชันหลัก (main function) ทำหน้าที่เรียกฟังก์ชันอื่น ๆ ในคลังโปรแกรมมาใช้งาน ฟังก์ชันหลักนี้มีลักษณะเป็น

   ```
   #include <robot.h>
   main()
   { อ่านไฟล์พารามิเตอร์แขนกล
     คำนวณโดยเรียกฟังก์ชันอื่น ๆ
   }
   ```

   ดังตัวอย่างของโปรแกรมแสดงในหน้าที่ 46

5. กำหนดพารามิเตอร์ของแขนกลในไฟล์ข้อมูล สมมติว่าชื่อ robot.txt

6. ทำการแปลและเชื่อมโยงไฟล์ demo.c ให้เป็นไฟล์ปฏิบัติการ (execute file)

   $ gcc demo.c -o demo.out -lm -lrobot

7. นำไฟล์ปฏิบัติการที่ได้มาใช้งาน ร่วมกับไฟล์พารามิเตอร์แขนกล

   $ ./demo.out robot.txt

ก.2 ไฟล์ข้อมูลพารามิเตอร์ของแขนกล

แขนกลที่ใช้กับคลังโปรแกรมนี้มีลักษณะของแขนเป็นท่อนๆ (link) เรียงต่อกัน โดยแต่ละท่อนเชื่อมต่อกันด้วยข้อต่อ (joint) ข้อต่อที่ใช้มี 2 ชนิด คือ ข้อต่อแบบหมุน (revolute) และแบบเลื่อน

Program n.1.1 demo.c

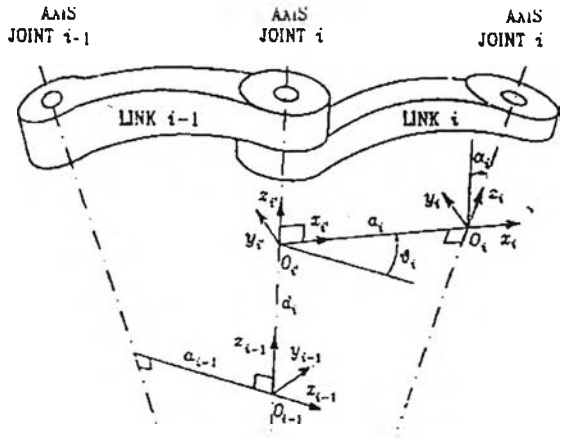
```
#include "robot.h"
main(int argc,char *argv[])
{   float t,tf,t1[MaxJoint],t2[MaxJoint];
int dir[MaxJoint],j;
Robot MK2,*prob;
real tau[MaxJoint]={0,0,0,0,0,0},otherq[MaxJoint];
real qi[]={0,0,0,0,0,0}, qf[]={0,0,0,0,0,0};
real ddqde[MaxJoint],dqde[MaxJoint],qde[MaxJoint];
real dqmax[]={0,Pi/2,Pi/2,Pi/2,Pi/2,2*Pi};   /* Max. Joint vel. (rad/s) */
real xyzpri[]={0,0.40,0.0,0.10,0,0};        /* initial position */
real xyzprf[]={0,0.0,0.60,0.10,0,0};        /*  final position */
real **TrMat;
if (readpara(argv[1],&MK2) == -1)     /* read parameter from file */
      exit(1);     prob = &MK2;
tf=1.5;    /* final time */
InverseKinematicMK2(&MK2,xyzprf,qf,otherq);  /*change XYZ pos. to an-
gles */
   InverseKinematicMK2(&MK2,xyzpri,qi,otherq);
 copy_rvec(qi,prob->q,MK2.total);     /*set robot pos. to initial pos.*/

 Trajectory(qi,qf,dqmax,tf,t1,t2,dir,MK2.total);
 copy_rvec(prob->q,qde,MK2.total);    /*initial value for qde, dqde*/
 copy_rvec(prob->dq,dqde,MK2.total);
for(j=1;j<=MK2.total;j++)   ddqde[j] = dir[j]*dqmax[j]/t1[j];

for(t=0;t<=tf;t=t+TimeSamp)
{ for(j=1;j<=MK2.total;j++)
  { if(t<t1[j])
ddqde[j] = dir[j]*dqmax[j]/t1[j];
   else
    { if(t<t2[j]+t1[j])  ddqde[j] = 0.0;
 else   ddqde[j] = -dir[j]*dqmax[j]/t1[j];
    } }  /* compute for desired trajectory */
 integrate(qde,dqde,ddqde,MK2.total,TimeSamp);
 InverseControl(&MK2,qde,dqde,ddqde,tau);    /* use  Inv Dyn Control*/
   rk4(prob,tau,TimeSamp,1); /* solve diff eq by rk4 */
}
freeRobot(&MK2);
}
```

ที่ (prismatic)

ในการกำหนดค่าพารามิเตอร์สำหรับแขนกล เริ่มต้นด้วย การกำหนดแกนพิกัดสำหรับท่อนแขนแต่ละท่อนโดยใช้วิธีของ Denavit-Hartenberg [18, 20] ซึ่งกำหนดให้แกนพิกัด 0 เป็นแกนอ้างอิง (base frame) ซึ่งเป็นแกนพิกัดที่อยู่กับที่ ส่วนแกนชุดที่ 1 จะเคลื่อนที่ตามแขนท่อนที่ 1 และกำหนดไปเรื่อยจนถึงแกนที่ n ดังรูปที่ ก.1 หลังจากนั้น พิจารณาให้แขนกลแต่ละท่อน เป็นแขนแบบแต่งเติม (augmented link) กล่าวคือ บนแขนกลท่อนที่ j มีมอเตอร์ตัวที่ $j+1$ อยู่



รูปที่ ก.1: การกำหนดแกนอ้างอิงตามวิธีของ Denavit-Hartenberg

ไฟล์ข้อมูลพารามิเตอร์ของแขนกลมีลักษณะดังนี้

- บรรทัดแรกเป็นจำนวนข้อต่อของแขนกล n
- บรรทัดที่สอง ถึง $n+1$ เป็นค่าตำแหน่ง q ความเร็ว $\dot{q}$ และความเร่ง $\ddot{q}$ เริ่มต้นของแต่ละข้อต่อ
- บรรทัดที่ $n+1$ ถึง $2n+1$ เป็นค่าพารามิเตอร์ของแขนกลแต่ละท่อน โดยในแต่ละแถวประกอบด้วย
  - a แทนระยะทางระหว่าง จุด O_i กับ $O_{i'}$
  - d เป็นพิกัดของ จุด $O_{i'}$ ในแกน z_{i-1}
  - α เป็นมุมระหว่างแกน z_{i-1} กับ z_i รอบแกน x_{i-1}
  - ϑ_i เป็นมุมระหว่างแกน x_{i-1} กับ x_i รอบแกน z_{i-1}
  - ชนิดของข้อต่อ ซึ่งกำหนดให้ 0 และ 1 แทนข้อต่อแบบหมุน (revolute) และเลื่อนที่ (prismatic)
  - มวล m ของท่อนแขนแต่งเติม
  - r_x
  - r_y

- r_z โดยที่ (r_x, r_y, r_z) เป็นเวกเตอร์ที่ชี้จาก จุด O_i ไปยัง จุดศูนย์กลางมวลของแขน เทียบกับพิกัดที่ i
- I_{xx}
- I_{yy}
- I_{zz}
- I_{xy}
- I_{yz}
- I_{xz} โดยที่ $\begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix}$ เป็นเมตริกซ์เทนเซอร์ความเฉื่อยของท่อนแขน เทียบกับแกนพิกัดที่ i
- I_m เป็นค่าความเฉื่อยของโรเตอร์ของมอเตอร์
- k_r อัตราส่วนลดของเกียร์
- z_x
- z_y
- z_z โดยที่ (z_x, z_y, z_z) เป็นเวกเตอร์หน่วย แสดงทิศทางของโรเตอร์ของมอเตอร์ตัวที่ j เทียบกับแกนพิกัดที่ $j-1$
- F_v เป็นค่าสัมประสิทธิความเสียดทานจากความหนืด (viscous friction)
- F_s เป็นค่าสัมประสิทธิความเสียดทานคูลอมบ์ (Coulomb friction)

- บรรทัดที่ $2n+1$ ถึง $2n+5$ เป็นเมตริกซ์การแปลงจากแกนพิกัดชุดสุดท้าย ไปยังแกนพิกัดของ end-effector ในกรณีที่ไม่ต้องการใช้ ให้กำหนดเมตริกซ์นี้เป็นเมตริกซ์เอกลักษณ์
- บรรทัดที่ $2n+6$ เป็นเวกเตอร์ของแรงภายนอกที่มากระทำแขนกลท่อนสุดท้าย
- บรรทัดที่ $2n+7$ เป็นเวกเตอร์ของโมเมนต์ภายนอกที่มากระทำแขนกลท่อนสุดท้าย
- บรรทัดที่ $2n+8$ เป็นเวกเตอร์ของแรงโน้มถ่วง

ตัวแปรที่ถูกนิยามขึ้นมาใหม่สำหรับคลังโปรแกรม ได้แก่ ตัวแปรชนิด real นั้น โดยทั่วไปกำหนดให้ตัวแปรนี้เป็น double เพื่อความถูกต้องในการคำนวณ แต่ถ้าผู้ใช้ต้องการประหยัดหน่วยความจำ ก็สามารถเปลี่ยนเป็นตัวแปรชนิด float ได้ง่าย โดยการเปลี่ยนนิยามในไฟล์ robot.h, ตัวแปรชนิด Para ซึ่งประกอบด้วยข้อมูลของแขนกลแต่ละท่อน โดยได้จากข้อมูลระหว่างบรรทัดที่ $n+1$ ถึง $2n+1$ และตัวแปรชนิด Robot ซึ่งประกอบด้วย ตัวแปรชนิด Para อยู่ภายใน และพารามิเตอร์อื่น ๆ ที่เหลือ

ชื่อฟังก์ชัน	หน้าที่
allocate_robot_para	ใช้จองพื้นที่หน่วยความจำสำหรับเก็บค่าพารามิเตอร์ของแขนกล
free_robot_para	ใช้คืนพื้นที่หน่วยความจำของตัวแปรที่เก็บค่าพารามิเตอร์ของแขนกล
freeRobot	ใช้คืนพื้นที่หน่วยความจำของตัวแปรที่เก็บข้อมูลแขนกลทั้งหมด
readpara	ใช้อ่านค่าพารามิเตอร์แขนกลจากไฟล์เก็บค่าลงในตัวแปรแขนกล
DirectKinematic	ใช้หาเมตริกซ์การแปลงระหว่างแกนพิกัดสองชุดใด ๆ
InverseKinematicMK2	ใช้หาตำแหน่ง q จากตัวแปรในปริภูมิการทำงาน ของแขนกลแบบ MK2
VelocityKinematic	ใช้หาเมตริกซ์จาโคเบียน $J(q)$
DirectDynamic	ใช้หาความเร่ง $\ddot{q}$ จากแรงบิด τ ที่ป้อนเข้า
InverseDynamic	ใช้หาแรงบิด τ ของแขนกลที่ตำแหน่ง ความเร็ว และความเร่งเป็น q $\dot{q}$ $\ddot{q}$
InertiaMatrix	ใช้หาเมตริกซ์ความเฉื่อย $B(q)$ ของแขนกล
selectTorque	ใช้หาค่าของแรงบิดที่เกิดจาก ความเฉื่อย หรือแรงโคริออลิส-แรงเข้าสู่ศูนย์กลาง หรือแรงโน้มถ่วง
InverseControl	ใช้หาสัญญาณควบคุมโดยใช้วิธีพลวัตผกผัน (Inverse Dynamic Control)

ตารางที่ ก.1: ฟังก์ชันสำหรับการคำนวณในส่วนของแขนกล

ก.3 วิธีการใช้งานฟังก์ชันภายในคลังโปรแกรม

ฟังก์ชันภาษา C ทั้งหมด แสดงอยู่ในตารางที่ ก.1-ก.3 สำหรับส่วนของวิธีการใช้งานแต่ละฟังก์ชันจะเรียงตามลำดับตัวอักษร

- void add_rvec(real ra[],real rb[],real rc[],int u);
  เวกเตอร์ rc เป็นผลบวกของเวกเตอร์ ra และ rb โดยที่ u เป็นขนาดของเวกเตอร์

- void add_rmat(real **A,real **B,real **C,int r,int c);
  เมตริกซ์ C เป็นผลบวกของเมตริกซ์ A และ B โดยมีขนาดเป็น r แถว และ c คอลัมน์

- int *allocate_integer_vector(int l,int u);
  ใช้จองพื้นที่ตัวแปรเวกเตอร์ที่เป็นเลขจำนวนเต็ม โดยที่ l และ u เป็นดรรชนีของสมาชิกตัวแรกและตัวสุดท้าย
  ตัวอย่าง: ใช้จองตัวแปรเวกเตอร์ vec ซึ่งมีสมาชิก 3 ตัว
  int *vec;
  vec = allocate_integer_vector(1,3);

- real **allocate_real_matrix(int lr,int ur,int lc,int uc);
  ใช้จองพื้นที่ตัวแปรเมตริกซ์ที่เป็นเลขค่าจริง โดยที่ lr, ur, lc และ uc เป็นดรรชนีของเมตริกซ์แถวแรก, แถวสุดท้าย, คอลัมน์แรก และคอลัมน์สุดท้าย
  ตัวอย่าง: ใช้จองตัวแปรเมตริกซ์ mat ขนาด 3 แถว 4 คอลัมน์

ชื่อฟังก์ชัน	หน้าที่
allocate_real_vector	ใช้จองพื้นที่หน่วยความจำสำหรับตัวแปรเวกเตอร์ที่เป็นเลขจริง
allocate_integer_vector	ใช้จองพื้นที่หน่วยความจำสำหรับตัวแปรเวกเตอร์ที่เป็นเลขจำนวนเต็ม
free_real_vector	ใช้คืนพื้นที่หน่วยความจำสำหรับตัวแปรเวกเตอร์ที่เป็นเลขจริง
free_integer_vector	ใช้คืนพื้นที่หน่วยความจำสำหรับตัวแปรเวกเตอร์ที่เป็นเลขจำนวนเต็ม
allocate_real_matrix	ใช้จองพื้นที่หน่วยความจำสำหรับตัวแปรเมตริกซ์ที่เป็นเลขจริง
free_real_matrix	ใช้คืนพื้นที่หน่วยความจำสำหรับตัวแปรเมตริกซ์ที่เป็นเลขจริง
copy_rvec	ใช้ทำสำเนาค่าของเวกเตอร์
add_rvec	ใช้หาผลบวกของเวกเตอร์
minus_rvec	ใช้หาผลต่างของเวกเตอร์
cross_rvec	ใช้หาผลคูณไขว้ของเวกเตอร์
dot_rvec	ใช้หาผลคูณจุดของเวกเตอร์
scalar_mul_rvec	ใช้หาผลคูณระหว่างเวกเตอร์กับสเกลาร์
mul_rmat_rvec	ใช้หาผลคูณระหว่างเมตริกซ์กับเวกเตอร์
print_rvec	ใช้แสดงค่าเวกเตอร์บนหน้าจอคอมพิวเตอร์
add_rmat	ใช้หาผลบวกของเมตริกซ์
copy_rmat	ใช้ทำสำเนาค่าของเมตริกซ์
mul_rmat	ใช้หาผลคูณของเมตริกซ์
scalar_mul_rmat	ใช้หาผลคูณระหว่างเมตริกซ์กับสเกลาร์
trace_rmat	ใช้หาผลบวกในแนวทแยงของเมตริกซ์
trans_rmat	ใช้หาเมตริกซ์สลับเปลี่ยน (transpose matrix)
inv_rmat	ใช้หาเมตริกซ์ผกผัน (inverse matrix)
print_rmat	ใช้แสดงค่าเมตริกซ์บนหน้าจอคอมพิวเตอร์

ตารางที่ ก.2: ฟังก์ชันสำหรับการจัดการเกี่ยวกับตัวแปรเวกเตอร์และเมตริกซ์

ชื่อฟังก์ชัน	หน้าที่
rk4	ใช้แก้สมการอนุพันธ์เพื่อหา q และ $\ddot{q}$ จากแรงบิดที่ป้อนเข้า
integrate	ใช้หา q_d และ $\dot{q}_d$ จาก $\ddot{q}_d$
print_file	ใช้เก็บค่าตัวแปรต่าง ๆ ลงไฟล์ข้อมูล
print_desire	ใช้เก็บค่าแนวการเคลื่อนที่ลงไฟล์ข้อมูล
Trajectory	ใช้ในการหาเวลา t_1 และ t_2 สำหรับสร้างแนวการเคลื่อนที่

ตารางที่ ก.3: ฟังก์ชันอื่นๆ

```
real **mat;
mat = allocate_real_matrix(1,3,1,4);
```

- real *allocate_real_vector(int l,int u);
  ใช้จองพื้นที่ตัวแปรเวกเตอร์ที่เป็นเลขค่าจริง โดยที่ l และ u เป็นดรรชนีของสมาชิกตัวแรกและตัวสุดท้าย

- Para *allocate_robot_para(int total);
  ใช้จองพื้นที่หน่วยความจำสำหรับตัวแปร Para โดยที่ total เป็นจำนวนข้อต่อทั้งหมดของแขนกล

- void copy_rmat(real **A,real **B,int r,int c);
  ใช้ทำสำเนาค่าของเมตริกซ์ A ลงในเมตริกซ์ B โดยมีขนาดเป็น r แถว และ c คอลัมน์

- void copy_rvec(real ra[],real rb[],int u);
  ใช้ทำสำเนาค่าของเวกเตอร์ ra ซึ่งมีขนาด u ลงในเวกเตอร์ rb

- void cross_rvec(real ra[],real rb[],real rc[]);
  เวกเตอร์ rc เป็นผลคูณไขว้ของเวกเตอร์ (cross product) ra และ rb ซึ่งเวกเตอร์ทุกตัวมีสมาชิกทั้งหมด 3 ตัว

- void DirectDynamic(real ddq[],Robot *prob,real tau[]);
  ใช้หาความเร่ง $\ddot{q}$ จากแรงบิด τ ที่ป้อนให้กับแขนกลซึ่งถูกชี้ตำแหน่งโดยตัวแปร prob

- void DirectKinematic(Robot *prob,real **TrMat,int m,int n);
  ใช้หาเมตริกซ์การแปลง TrMat ระหว่างแกนพิกัดที่ m ไปยัง n ($\boldsymbol{A}_m^n$) ของตัวแปรแขนกล โดยที่ m มีค่าน้อยกว่า n

  ตัวอย่าง หา $\boldsymbol{A}_0^5$ ของตัวแปรแขนกล MK

  ```
  Robot MK;
  DirectKinematic(&MK,TrMat,0,5);
  ```

- real dot_rvec(real ra[],real rb[],int u);
  ฟังก์ชันนี้คืนค่าเป็นผลคูณจุด (dot product) ระหว่างเวกเตอร์ ra และ rb ซึ่งมีขนาดเป็น u

- void free_integer_vector(int *v,int l);
  ใช้คืนพื้นที่หน่วยความจำของตัวแปรเวกเตอร์ชนิดที่เป็นเลขจำนวนเต็ม v โดยที่ l เป็นดรรชนีของสมาชิกตัวแรก

- void free_real_matrix(real **m,int lr,int ur,int lc);
  ใช้คืนพื้นที่หน่วยความจำของตัวแปรเมตริกชนิดที่เป็นเลขจำนวนจริง m โดยที่ lr, ur และ lc เป็นดรรชนีของเมตริกซ์แถว, แถวสุดท้าย และคอลัมน์แรก

- void free_real_vector(real *v,int l);
  ใช้คืนพื้นที่หน่วยความจำของตัวแปรเวกเตอร์ชนิดที่เป็นเลขจำนวนจริง v โดยที่ l เป็นดรรชนีของสมาชิกตัวแรก

- void free_robot_para(Para *ppara);
  ใช้คืนพื้นที่หน่วยความจำของตัวแปร Para ที่ชี้ตำแหน่งที่อยู่โดย ppara

- void freeRobot(Robot *prob);
  ใช้คืนพื้นที่หน่วยความจำของตัวแปร Robot ที่ชี้ตำแหน่งที่อยู่โดย prob

- void integrate(real q_des[],real dq_des[],real ddq_des[],int total,float h);
  ใช้หาเวกเตอร์ของตำแหน่ง q_des และความเร็ว dq_des ของแนวการเคลื่อนที่ที่ต้องการ จากความเร่ง ddq_des โดยที่ total เป็นจำนวนข้อต่อ และ h บอกช่วงเวลาสุ่ม

- void inv_rmat(real **a,real **b,int n);
  ใช้หาเมตริกซ์ b ซึ่งเป็นเมตริกซ์ผกผันของเมตริกซ์ a โดยที่ n เป็นจำนวนแถวหรือคอลัมน์ของเมตริกซ์

- void InverseControl(Robot *prob,real q_des[],real dq_des[],real ddq_des[],real tau[]);
  ใช้หาสัญญาณควบคุม tau โดยวิธีพลวัตผกผัน จากแนวการเคลื่อนที่ q_des, dq_des และ ddq_des

- void InverseKinematicMK2(Robot *prob,real xyzpr[],real q1[],real q2[]);
  ใช้สำหรับแขนกลแบบ MK2 ซึ่งมี 5 ข้อต่อแบบหมุนในการแปลงตำแหน่งจากปริภูมิการทำงาน xyzpr ประกอบด้วย พิกัดในแกน xyz และมุม pitch กับ row ให้เป็นตำแหน่งในปริภูมิข้อต่อ ซึ่งมีค่าเป็นไปได้ 2 ค่า คือ q1 และ q2 [1]

- void InertiaMatrix(Robot *prob,real **MatB);
  ใช้หาเมตริกซ์ความเฉื่อย MatB ของแขนกล prob โดยใช้ค่าตำแหน่ง q ซึ่งเก็บภายในตัวแปรแขนกลนี้

- void InverseDynamic(Robot *prob,real tau[]);
  ใช้หาแรงบิด tau ของแขนกล prob ขณะที่มีค่าตำแหน่ง ความเร็ว ความเร่ง เป็น $q\ \dot{q}\ \ddot{q}$

- void minus_rvec(real ra[],real rb[],real rc[],int u);
  เวกเตอร์ rc เป็นผลต่างระหว่างเวกเตอร์ ra และ rb ซึ่งมีขนาดเป็น u

- void mul_rmat(real **A,real **B,real **C,int ra,int ca,int cb);
  เมตริกซ์ C เป็นผลคูณระหว่างเมตริกซ์ A ซึ่งมีขนาด ra แถว และ ca คอลัมน์ กับเมตริกซ์ B ซึ่งมีขนาด ca แถว และ cb คอลัมน์

- void mul_rmat_rvec(real **A,real rb[],real rc[],int r,int c);
  เวกเตอร์ rc เป็นผลคูณระหว่างเมตริกซ์ A กับเวกเตอร์ rb โดยเมตริกซ์ A มีขนาดเป็น r แถว และ c คอลัมน์

- void print_desire(float t,real q[],real dq[],real ddq[],FILE *ftraj,int total);
  ใช้เก็บข้อมูลการเคลื่อนที่ในไฟล์ ftraj โดยประกอบด้วย เวลา t, ตำแหน่ง q, ความเร็ว dq, ความเร่ง ddq และ total เป็นขนาดของเวกเตอร์

- void print_file(Robot *prob,float t,real tau[],FILE *fout);
  ใช้เก็บข้อมูลของการจำลองระบบ ลงในไฟล์ fout โดยที่ คอลัมน์แรกเป็นเวลา t คอลัมน์ถัดมาเป็นค่าของแรงบิด tau ตำแหน่ง ความเร็ว และความเร่ง ของข้อต่อที่ 1 ซึ่งอยู่ในตัวแปร prob และตามด้วยข้อมูลของข้อต่อที่ 2 จนถึงข้อต่อสุดท้าย และท้ายสุดเป็นตำแหน่งปลายแขนในปริภูมิการทำงาน ได้แก่ พิกัด xyz มุม Roll Pitch Yaw

- void print_rmat(real **A,int r,int c,char string[]);
  ใช้แสดงค่าเมตริกซ์ A บนหน้าจอคอมพิวเตอร์ โดยเมตริกซ์มีขนาด r แถว และ c คอลัมน์ และ string เป็นข้อความสำหรับใช้อธิบายเมตริกซ์นั้น

- void print_rvec(real *a,int r,char string[]);
  ใช้แสดงค่าเวกเตอร์ a บนหน้าจอคอมพิวเตอร์ โดยมีจำนวนสมาชิก r ตัว และ string เป็นข้อความสำหรับใช้อธิบายเวกเตอร์นั้น

- int readpara(const char filename[],Robot *prob);
  ใช้อ่านไฟล์พารามิเตอร์แขนกลที่อยู่ใน filename เพื่อเก็บในตัวแปร prob โดยปกติฟังก์ชันจะคืนค่า 0 แต่ในกรณีที่หาไฟล์ไม่เจอ จะคืนค่าเป็น -1

- void rk4(Robot *prob,real torq[],real samp,int ratio);
  ใช้แก้สมการอนุพันธ์โดยวิธี Runge-Kutta อันดับ 4 เพื่อหาตำแหน่งและความเร็วของแขนกล prob จากแรงบิด torq ที่ป้อนเข้าไป โดยที่ samp เป็นช่วงเวลาสุ่ม และ อัตราส่วนระหว่าง samp กับ ratio เป็นขนาดขั้น (step size) ในการแก้สมการ

  ตัวอย่าง: หาค่าตำแหน่งและความเร็วของแขนกล จากแรงบิดที่ป้อนเข้า
  โดยมี ช่วงเวลาสุ่มเป็น 0.001 วินาที และ ขนาดขั้นเป็น 0.0001 วินาที

```
Robot MK2;
real torq[3] = {0, 10.0, 5.0};
rk4(&MK2, torq, 0.001, 10);
```

- void scalar_mul_rmat(real **A,real **C,int r,int c,real s);
  เมตริกซ์ C เป็นผลคูณระหว่างเมตริกซ์ A ซึ่งมีขนาด r แถว และ c คอลัมน์ กับสเกลาร์ s

- void scalar_mul_rvec(real ra[],real rb[],int u,real s);
  เวกเตอร์ rb เป็นผลคูณระหว่างเวกเตอร์ ra ซึ่งมีขนาดเป็น u กับสเกลาร์ s

- void selectTorque(Robot *prob,real taus[],char opt);
  ใช้หาแรงบิดที่กระทำแขนกล prob แล้วแต่ประเภทของแรง โดยถ้า opt เป็นตัว i เป็นแรงที่เกิดจากเมตริกซ์ความเฉื่อย, ตัว c เป็นผลของแรงโคริออลิส-แรงสู่ศูนย์กลาง, ตัว g เป็นผลของแรงโน้มถ่วง, และ ตัว a เป็นผลของแรงโคริออลิส-แรงสู่ศูนย์กลาง รวมกับ ผลของแรงโน้มถ่วง

- void Trajectory(real qi[],real qf[],real dqmax[],real tf,real t1[],real t2[],int dir[],int total);
  ใช้หาเวลา t1 และ t2 สำหรับการสร้างแนวทางการเคลื่อนที่ ซึ่งเริ่มจากตำแหน่ง qi ไปยัง qf มีความเร็วเชิงมุมสูงสุดเป็น dqmax โดยที่ ช่วงเวลา 0–t1 เป็นช่วงที่ให้ความเร่งเชิงมุมเป็นค่าบวกคงที่ ช่วงเวลา t1–t2 เป็นช่วงที่ความเร็วเชิงมุมคงที่ (ความเร่งมีค่าเป็นศูนย์) และในช่วงสุดท้าย t2–tf เป็นช่วงที่ความเร่งเชิงมุมเป็นค่าลบคงที่ ส่วน dir เป็นตัวบอกทิศทางการหมุน โดยมีค่าเป็น 1 เมื่อ qi น้อยกว่า qf และเป็น -1 เมื่อ qi มากกว่า qf

  ตัวอย่าง: สำหรับแนวการเคลื่อนที่ ซึ่งมีตำแหน่งเริ่มต้น กับความเร็วสูงสุด เป็น 0 rad
  กับ Pi/4 rad/s ทั้งคู่ ตำแหน่งสุดท้ายเป็น Pi/2 และ $-$Pi/2
  และใช้เวลาในการเคลื่อนที่ทั้งหมด 3 วินาที

```
real qi[]={0, 0, 0}, qf[]= {0, Pi/2, −Pi/2}, dqmax = {0, Pi/4, Pi/4};
real t1[3], t2[3], tf= 3;
int dir[3];
Trajectory(qi, qf, dqmax, tf, t1, t2, dir, 2);
```

  โดยจะได้ t1 มีค่าเป็น {0,1,1}, t2 มีค่าเป็น {0,2,2} และ dir มีค่าเป็น {0,1,-1}

- real trace_rmat(real **A,int u);
  ฟังก์ชันนี้คืนค่าเป็นผลบวกในแนวทแยงมุมของเมตริกซ์ A ซึ่งเป็นเมตริกซ์จัตุรัส และมีขนาด u แถว

- void trans_rmat(rcal **A,real **C,int r,int c);
  เมตริกซ์ C ที่ได้เป็นเมตริกซ์สลับเปลี่ยน (transpose) ของเมตริกซ์ A ซึ่งมีขนาด r แถว และ c คอลัมน์

- void VelocityKinematic(Robot *prob,real **Jaco);
  ใช้หาเมตริกซ์จาโคเบียน Jaco ของแขนกล prob ซึ่งอยู่ในตำแหน่ง q

ภาคผนวก ข

ส่วนติดต่อกับผู้ใช้ทางภาพกราฟิก

ข.1 การสร้างและติดตั้งส่วนติดต่อกับผู้ใช้ทางภาพกราฟิก

การสร้าง GUI โดยใช้ GTK ในการเขียนโปรแกรมทั้งหมดนั้นมีข้อเสีย 2 ประการ ประการแรก การสร้าง GUI นั้นมีขั้นตอนที่ซ้ำซ้อนอยู่ ตัวอย่างเช่น การสร้างวัตถุ (object) ได้แก่ หน้าต่าง (window), ปุ่ม (button), ป้าย (label) ที่มีลักษณะคล้ายกัน ประการที่สองผู้เขียนโปรแกรมไม่เห็นหน้าตาของ GUI ที่กำลังเขียนอยู่ จนกว่าจะทำการแปลโปรแกรมให้เป็นไฟล์ที่ทำงานได้ก่อน ซึ่งลักษณะของ GUI ที่ได้อาจไม่ตรงตามความต้องการทำให้ผู้เขียนต้องเสียเวลาในการแก้ไขและแปลโปรแกรมหลายครั้ง โดยเฉพาะผู้เขียนที่ยังไม่คุ้นเคย หรือไม่รู้ฟังก์ชันใน GTK มากพอ

เพื่อลดภาระข้างต้นของผู้เขียน ได้มีผู้สร้างเครื่องมือช่วย เช่น GLADE ซึ่งมีหน้าที่ คือ สร้าง GUI ประเภทได้อย่างที่เห็น (what you is what you get: WYSIWYG) ทำให้ผู้เขียนสามารถแก้ไข GUI ให้ได้ลักษณะ หน้าตา ตรงตามความต้องการ แล้วให้เครื่องมือเหล่านั้นสร้างรหัสต้นฉบับเพื่อใช้แก้ไขโปรแกรม และเขียนฟังก์ชันเพิ่มเติมในส่วนของการทำงานของ GUI ในภายหลัง อย่างไรก็ตาม เครื่องมือเหล่านั้นมีข้อเสีย คือ รหัสต้นฉบับที่ได้จะไม่กะทัดรัดเหมือนการเขียนเองและ GUI ที่สร้างขึ้นมีความสามารถจำกัด เช่น ไม่สามารถสร้างหน้าต่างที่มีจำนวนปุ่มเปลี่ยนแปลงตามจำนวนข้อต่อของแขนกล

สำหรับ GUI ของแขนกลนั้น ขั้นตอนแรก ใช้ GLADE ในการสร้าง GUI เพื่อให้ได้หน้าตาตามต้องการ แล้วจึงทำการแก้ไขรหัสต้นฉบับที่ได้จาก GLADE โดยเขียนฟังก์ชันเพิ่มเติมสำหรับหน้าที่ของปุ่มต่าง ๆ บน GUI, สำหรับแสดงผลเป็นรูปเคลื่อนไหว 3 มิติ และสำหรับแสดงกราฟของสัญญาณ

ก่อนที่จะติดตั้งส่วนของ GUI จำเป็นต้องติดตั้งคลังโปรแกรมให้เรียบร้อยก่อน จากนั้นใช้ Makefile สำหรับแปลและเชื่อมโยงไฟล์ต้นฉบับต่าง ๆ ให้กลายเป็นไฟล์ปฏิบัติการ โดยผู้ใช้เพียงแค่พิมพ์ make ที่บรรทัดคำสั่ง ก็จะได้ไฟล์ชื่อ simrobot สำหรับใช้งาน

ข.2 การใช้งานในส่วนติดต่อกับผู้ใช้ทางภาพกราฟิก

ขั้นตอนคร่าว ๆ สำหรับการใช้งานในส่วนของ GUI เป็นดังนี้

1. โหลดไฟล์พารามิเตอร์แขนกลของแขนกลที่ต้องการ โดยคลิกปุ่ม Load หรือพิมพ์ไดเร็กทอรีและชื่อไฟล์ ในช่องรับเข้า
2. เลือกไฟล์สำหรับเก็บข้อมูลการเคลื่อนที่ของแขนกล โดยคลิกปุ่ม Save หรือพิมพ์ไดเร็กทอรีและชื่อไฟล์ ในช่องรับเข้า

3. เลือกแนวทางการเคลื่อนที่ โดยการคลิกที่ปุ่ม Trajectory ซึ่งจะทำให้ได้ หน้าต่างสำหรับป้อนค่าตำแหน่งเริ่มต้น, ตำแหน่งสุดท้าย, ความเร็วสูงสุด ของแต่ละข้อต่อ และเวลาสุดท้าย สำหรับพารามิเตอร์สามตัวแรก ผู้ใช้ต้องป้อนในลักษณะของเวกเตอร์
ตัวอย่างเช่น ต้องการกำหนดตำแหน่งเริ่มต้นเป็น $0,\ \pi/2,\ -\pi/2,\ \pi/2+\pi/4,\ 3\pi/2$ rad ของแขนกลที่มี 5 ข้อต่อ ก็ต้องพิมพ์ "[0, Pi/2, −Pi/2, Pi/2 + Pi/4, 3 ∗ Pi/2]"

ในส่วนของตัว PID ไม่ได้ควบคุมให้เคลื่อนที่ตามแนวทางที่กำหนด แต่ต้องการให้ข้อต่อหมุนไปยังตำแหน่งสุดท้ายที่ต้องการเลย กล่าวคือ ความคลาดเคลื่อนของมุมได้จาก ผลต่างของตำแหน่งสุดท้ายกับตำแหน่งปัจจุบันของแต่ละข้อต่อ

4. เลือกตัวควบคุมแบบพลวัตผกผัน, โครงสร้างแปรผันได้ หรือ PID แบบใดแบบหนึ่ง โดยการคลิกปุ่ม INV DYN, VSC หรือ PID ซึ่งจะได้หน้าต่างสำหรับป้อนพารามิเตอร์

สำหรับตัวควบคุม PID ให้ป้อนค่าอัตราขยาย $\boldsymbol{K}_p$ และ $\boldsymbol{K}_d$ ในลักษณะของตัวแปรเวกเตอร์

สำหรับตัวควบคุมแบบพลวัตผกผัน และโครงสร้างแปรผันได้ พารามิเตอร์ได้แก่ $\boldsymbol{K}_p$, $\boldsymbol{K}_d$, $\boldsymbol{P}$ และ $\boldsymbol{Q}$ ให้ป้อนในลักษณะของเมตริกซ์
ตัวอย่างเช่น ต้องการป้อนเมตริกซ์ $\begin{bmatrix} 10 & 1 \\ 2 & 20 \end{bmatrix}$ ก็ต้องพิมพ์ "[10, 1; 2, 20]"
นอกจากนั้นในกรณีตัวอย่างต่อไปนี้ผู้ใช้สามารถพิมพ์ย่อได้

- ต้องการป้อนเมตริกซ์ที่เป็นเมตริกซ์ทแยง และมีสมาชิกเท่ากับ 25 สำหรับทุกข้อต่อ ก็สามารถพิมพ์ย่อเป็น "M25"
- สำหรับการควบคุมแบบโครงสร้างแปรผันได้ ในกรณีที่ $\boldsymbol{K}_p$, $\boldsymbol{K}_d$ เป็นเมตริกซ์ทแยง และมีสมาชิกในแต่ละเมตริกซ์เท่ากัน และเมตริกซ์ $\boldsymbol{P}$ มีลักษณะเป็น $\begin{bmatrix} a*\boldsymbol{I}_n & b*\boldsymbol{I}_n \\ b*\boldsymbol{I}_n & c*\boldsymbol{I}_n \end{bmatrix}$ โดยที่ a, b, c เป็นค่าตัวเลขใดๆ และ n เป็นจำนวนข้อต่อของแขนกล ผู้ใช้ก็จะสามารถพิมพ์ย่อสำหรับเมตริกซ์ $\boldsymbol{P}$ เป็น "P [a, b; b, c]" และสำหรับ $\boldsymbol{Q}$ พิมพ์แค่ "Q" เท่านั้น ส่วนในกรณีอื่นผู้ใช้ต้องไปแก้สมการ (2.25) มาจากโปรแกรมอื่นเอง

5. เริ่มการจำลองระบบ โดยคลิกปุ่ม START ซึ่งในขณะจำลองอยู่ ผู้ใช้สามารถหมุน, เลื่อนที่ และขยายขนาดภาพแขนกลได้ นอกจากนั้น อาจให้โปรแกรมหยุดชั่วคราว โดยคลิกปุ่ม PAUSE ซึ่งถ้าต้องการจำลองระบบต่อ ให้คลิกปุ่มนี้ซ้ำ ส่วนปุ่ม STOP ใช้สำหรับหยุดการจำลองระบบก่อนถึงเวลาสุดท้าย

6. หลังจากทำการจำลองระบบเสร็จแล้ว ผู้ใช้สามารถดูกราฟของสัญญาณตำแหน่ง ความเร็ว และแรงบิดของแต่ละข้อต่อได้ โดยคลิกปุ่ม Graph ซึ่งจะทำให้ได้ หน้าต่างสำหรับเลือกชนิดของ

สัญญาณที่ต้องการแสดงส่วนการแสดงมุม Roll, Pitch และ Yaw นั้น ขึ้นอยู่กับชนิดของแขนกล ผู้ใช้อาจต้องแก้ไขในไฟล์ต้นฉบับเอง

7. ในการออกจากโปรแกรม ให้คลิกปุ่ม EXIT และปุ่ม YES เพื่อยืนยันว่าต้องการออกจากโปรแกรมจริง

ข้อระมัดระวังในการใช้งาน ได้แก่

- เมื่อป้อนข้อมูลหรือค่าตัวเลขต่าง ๆ ในช่องรับเข้า (entry) ผู้ใช้ต้องกดปุ่ม enter ด้วย เพื่อกระตุ้นให้ไปเรียกฟังก์ชันสำหรับเปลี่ยนแปลงค่าข้อมูล ซึ่งโปรแกรมจะทำการแสดงค่าใหม่บนจอคอมพิวเตอร์
- ควรป้อนเมตริกซ์และเวกเตอร์ให้มีขนาดถูกต้อง และตรวจสอบค่าที่ได้บนจอคอมพิวเตอร์ให้ถูกต้องด้วย เพื่อป้องกันความผิดปกติของโปรแกรมที่อาจเกิดขึ้นได้
- ป้อนให้ เมตริกซ์ $\boldsymbol{K}_p$, $\boldsymbol{K}_d$, $\boldsymbol{P}$ และ $\boldsymbol{Q}$ เป็นเมตริกซ์บวกแน่นอนเสมอ

ประวัติผู้วิจัย

นายกมล ลิมธัญญกูล เกิดวันที่ 31 ตุลาคม พ.ศ. 2520 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาปริญญาตรีวิศวกรรมศาสตรบัณฑิต ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2540 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต ที่จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2541 โดยได้รับทุนอุดหนุนการศึกษาจากโครงการศิษย์ก้นกุฏิ ภาควิชาวิศวกรรมไฟฟ้า