# 6.095 Humanoid Robotics Competition

Massachusetts Institute of Technology

## Library Manual

KINEMATICS

```
void leg_ik(float* coords, float* angles)
```
requires: `float coords[6] ={x1,y1,z1,x2,y2,z2}`
modifies: `float angles[DOF]`

Will calculate the angles of the legs for the given coordinates.
Will only modify the necessary angles on the angles array. For instance, it will not change the angle values for the arms.

```
void arm_ik(float* coords, float* result_angles)
```
requires: `float coords[6] ={x1,y1,z1,x2,y2,z2}`
modifies: `float angles[DOF]`

Same as leg_ik, for arms.

I/O

```
void printNumber(char* string, float num)
```
requires: `char* string, float num`

Will print the number. For instance `printNumber("my_variable", my_variable)` would display `my_variable 11.3`

```
void printList(char* string, int* list, int size)
```
requires: `char* string, int list[size], int size`

Will print a list of integers.

```
void printListf(char* string, float* list, int size)
```
requires: `char* string, float list[size], int size`

Will print a list of floats.

```
void printString(char* string)
```
requires: `char* string`

Will print the string.

```
int user_input_int()
```
returns: the integer (positive or negative) input by the user.

```
int user_input_bool()
```
returns: 0 if the user pressed N and 1 if the user pressed Y

SERVO CONTROL

`void append_new_pose(float* angles, float time)`
requires: `float angles[DOF], float time`
Will append this pose to the queue. This function clones the content of the given array of angles, so it may be modified without worry later.

`int is_executing_motion()`
returns: 1 if a motion is being executed, 0 otherwise.

`int empty_spaces_in_queue()`
returns: the number of empty spaces in the queue. The queue has room for 200 poses when it is empty.

`void remove_all_motions()`
Clears the queue and stops the current motion.

`void wait_for_motion_to_finish()`
Will not return until there are no motions being executed.

`void get_standing_position(float* angles)`
modifies: `float angles[DOF]`
Will set the angles to the angles of the home position.

`void append_recorded_motion(int motion_index, float duration1)`
requires: `int motion_index, float duration1`
Will append the recorded motion corresponding to this index to the queue of poses. The duration given is the time the robot should take to move from the current pose to the first pose of the motion.

MATH

`int power(int base, int exp)`
requires: `int base, int exp`
returns: base ^ exp

```
float cosf(float rad_angle)
float sinf(float rad_angle)
float tanf(float rad_angle)
float acosf(float value)
float asinf(float value)
float atanf(float value)
float sqrtf(float value)
```

TOOLS

`void subListf(float* sub, float* list, int ini, int end)`
requires: `float* list, int ini, int end`
modifies: `float* sub`
Will copy the values between the ini index and the end index from the list array to the sub array.

`void copyListf(float* target, float* list, int size)`
requires: `float list[size], int size`
returns: `float target[size]`
Will copy the content of the array `list` into the array `target`. Both should be of size `size`.

`void fillf(float* lst, float val, int size)`
requires: `float lst[size], float val, int size`
Will fill the array `lst` with the given `val`.

`void fill(int* lst, int val, int size)`
requires: `int lst[size], int val, int size`
Will fill the array `lst` with the given `val`.

`void addListsf(float* child, float* parent, int len)`
requires: `float child[len]; float parent[len]; int len`
Will add child and parent element-by-element and return the result in child.

`float absolute(float val)`
requires: `float val`
returns: float, the absolute value of `val`

`float equalf(float val1, float val2)`
requires: `float val1, float val2`
returns: 1 if the two floats are equal, 0 otherwise.

`int sign(float val)`
requires: `float val`
returns: 1 of val is positive, -1 of val is negative

`float ptToPtDistance(float* p1, float* p2, int size)`
requires: `float p1[size], float p2[size], size`
returns: the norm of the vector between the two points.

`float max(float one, float two)`
requires: `float one, float two`
returns: the maximum value between the two.

CONTROLLER INPUT

**Warning: contrary to conventions, the value 1 means the button is not pressed and the value 0 means the button is being pressed.**

```
unsigned char   DSC2_dat.bit.select     // select
unsigned char   DSC2_dat.bit.start      // start
unsigned char   DSC2_dat.bit.up         // L2 button
unsigned char   DSC2_dat.bit.right      // R2 button
unsigned char   DSC2_dat.bit.down       // L1 button
unsigned char   DSC2_dat.bit.left       // R1 button
unsigned char   DSC2_dat.bit.L2         // L2 button
unsigned char   DSC2_dat.bit.R2         // R2 button
unsigned char   DSC2_dat.bit.L1         // L1 button
unsigned char   DSC2_dat.bit.R1         // R1 button
unsigned char   DSC2_dat.bit.triangle   // triangle button
unsigned char   DSC2_dat.bit.circle     // circle button
unsigned char   DSC2_dat.bit.X          // X button
unsigned char   DSC2_dat.bit.rectangle  // rectangle button

unsigned char   DSC2_dat.bit.r_a_lr;    // right analog stick L-R
unsigned char   DSC2_dat.bit.r_a_ud;    // right analog stick U-D
unsigned char   DSC2_dat.bit.l_a_lr;    // right analog stick L-R
unsigned char   DSC2_dat.bit.l_a_ud;    // right analog stick U-D
```