

MANIPULAÇÃO DE UM BRAÇO ROBÓTICO POR MOVIMENTOS DO BRAÇO E DA MÃO HUMANA USANDO SENSORES INERCIAIS

Pedro Henrique Ortiz Costa¹; Gustavo Santana Bastos²; Cleiton Silvano Goulart³; Leandro Aureliano da Silva⁴; Antonio Carlos Lemos Júnior⁵

^{1,2,3,4,5} Faculdade de Talentos Humanos - FACTHUS, Uberaba (MG), Brasil

pedroh.ortiz@hotmail.com, gustavo.gsbastos@hotmail.com, cleiton.goulart@facthus.edu.br, lasilva@facthus.edu.br, acjunior@facthus.edu.br

RESUMO: Atualmente, existe uma infinidade de modelos de manipuladores robóticos que contam com variados métodos de operação em que alguns deles exigem precisão e coordenação motora do operador. Esse artigo visou o desenvolvimento de uma metodologia de controle de um braço robótico por movimentos gestuais com a utilização de sensores inerciais MEMS. Tais sensores dispõem de IMU que afere os valores de giroscópio e de acelerômetro da placa de silício. O presente trabalho, ainda, utilizou um recurso chamado de *Digital Motion Processor* (DMP) integrado ao chip do MPU6050 e foi realizada uma comparação entre a variação dos valores do sensor com e sem o uso do DMP. Os resultados permitiram concluir que a inclusão do DMP à programação de controle diminuiu os ruídos e a instabilidade dos dados extraídos do MPU6050, o que garantiu ao braço robótico movimentos mais suaves e precisos.

PALAVRAS CHAVE: DMP, Inerciais; Movimentos, Robótico, Sensores.

MANIPULATION OF A ROBOTIC ARM THROUGH MOVEMENTS OF THE HUMAN ARM AND HAND USING INERTIAL SENSORS

ABSTRACT: Currently, there is an infinity of robotic manipulators models that have several operating methods in which some of them requires precision and motor coordination from the operator. This article aimed the development of a robotic arm control methodology through gestural movements using MEMS inertial sensors. Such sensors have IMU that measures the accelerometer and the gyroscope values of the silicon board. The present paper still used a feature called *Digital Motion Processor* (DMP) integrated to the MPU6050 chip and a comparison was made between the variation of the sensor values with and without the DMP. The results allowed to conclude that the inclusion of the DMP to the control programming reduced the noises and the instability of the data extracted from the MPU6050, which guaranteed the robotic arm smoother and precise movements.

KEYWORDS: DMP, Inertial, movements, Robotic, Sensors.

INTRODUÇÃO

Segundo o Dr. Andreas Bauer (BAUER, 2019), presidente do Comitê de Fornecedores de Robôs Industriais da *International Federation of Robotics* (IFR), a expectativa é que mais de 2 milhões de unidades de novos robôs serão instalados em indústrias ao redor do mundo somente do ano de 2020 a 2022. Esse crescente protagonismo de robôs no atual cenário tecnológico possibilitou uma infinidade de aplicações em que são explorados diversos métodos de controle de manipuladores robóticos. A utilização de sensores inerciais, produzidos a partir da revolucionária tecnologia *Micro Electro Mechanical Systems* - sistemas microeletromecânicos (MEMS), na manipulação de braços robóticos pode ser uma metodologia bastante promissora em teleoperações precisas que requerem comandos intuitivos reproduzidos por movimentos do próprio corpo humano, como os braços e as mãos. Dentre as quais se enquadram operações industriais, operações remotas em áreas insalubres, cirurgias minimamente invasivas e recuperação submarina (SEKHAR et al., 2012).

A captação de movimentos do corpo humano para controle de braços robóticos pode ser, também, feita por câmeras que, através de sensores visuais, detectam esses movimentos e enviam os dados obtidos para os atuadores. Entretanto e com base nos resultados obtidos por Torquato (2017), a área de alcance limitada da câmera e a interferência de cores e luminosidade que o ambiente pode gerar são algumas das restrições e dos problemas desse sistema. Tais limitações não são encontradas em sensores inerciais, fazendo, assim, com que esses possuam a flexibilidade de captar movimentos em diferentes tipos de ambientes, além de prover dados mais confiáveis e mais compatíveis com o movimento executado por estar acoplado junto ao operador (FANG et al., 2017). No entanto, os valores “crus” do giroscópio e do acelerômetro medidos por sensores inerciais, também chamados de *Inertial Measurement Unit* (IMU), possuem bastante ruídos e distúrbios magnéticos, resultando em dados com erros cumulativos (FIELD et al., 2011).

De acordo com César (2005), a robótica apresenta-se como um campo da engenharia cada vez mais abrangente e

multidisciplinar, o que faz essa ciência ter interessantes aplicações educacionais. Ademais, a ideia de fazer com que braços robóticos reproduzam movimentos corporais evidencia uma relação humana mútua e cooperativa com essa inovadora tecnologia. Ainda, o método de controle intuitivo e adaptativo de manipuladores estudado no presente trabalho pode ter aplicações de grande relevância em áreas fundamentais, como a medicina e em operações industriais precisas e específicas.

Neisy Amparo Escobar Forhan relata que a tecnologia MEMS em sensores inerciais é amplamente usada em sistemas mecânicos robóticos. Os acelerômetros estabelecem as oscilações do centro de massa do veículo, enquanto os giroscópios estabelecem as velocidades angulares do corpo em um eixo fixo. Porém, medir a rotação de um corpo rígido ou de um veículo com precisão requer um mínimo grau de erro nos resultados aferidos (FORHAN, 2010).

Para aplicações complexas e em interações com os usuários poderão ser necessários outros comandos gestuais. É importante que o reconhecimento seja ágil, em tempo real e o mais simples possível. Algumas estratégias podem ser empregadas neste problema. Primeiro é necessário representar os gestos através de, por exemplo, *feature vectors*. Os vetores podem conter dados relativos às posições (x, y, z) e orientação (*pitch*, *yaw*, *roll*) da mão assim como a posição e orientação das articulações suportados pelo sensor em cada movimento (VARUM, 2015).

Tendo isso em vista, esse artigo desenvolveu um método de controle de um protótipo de manipulador robótico que atuará baseado em 3 movimentos diferentes de acordo com movimentos realizados pelo braço e pela mão do operador com a utilização de sensores inerciais MEMS. Ainda, foi usado um recurso contido no chip dos sensores chamado de *Digital Motion Processor* (DMP) e, a partir disso, buscou-se a realização de uma comparação da movimentação do manipulador com e sem a utilização desse processador integrado.

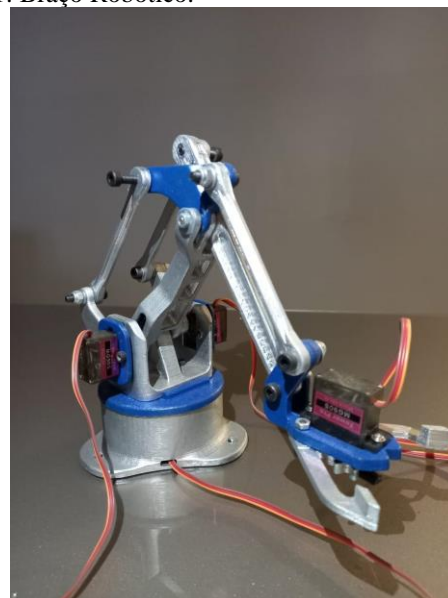
MATERIAL E MÉTODOS

Para laboração do referido estudo, foi utilizado um braço mecânico do modelo Mark1, cujas peças foram impressas em plástico PCL e, com a finalidade de controlá-lo, será acoplado um sensor MPU6050 de acelerômetro e giroscópio com 6 graus de liberdade à parte superior de uma luva que calçará a mão e outro mesmo sensor será disposto no antebraço do operador.

O braço robótico utilizado (Fig. 1) foi montado sobre uma base fixa e com uma estrutura de elos e juntas que concedem a ele 3 graus de liberdade (GDL). Além disso, foram empregados 4 micros servomotores como atuadores, tais tipos de motores possuem uma caixa de engrenagem que acumula relações de transmissões, permitindo um controle preciso no eixo de saída com uma rotação contínua de, aproximadamente 180°. O modelo dos servomotores usados nesse projeto foi o MG90S Tower Pro que se destaca pela

construção das engrenagens em metal, o que propicia movimentos ainda mais precisos e confiáveis, além de garantir mais força ao sistema (FILIPEFLOP, 2020).

Figura 1: Braço Robótico.



Fonte: Os autores, 2020.

Na aparelhagem de controle (Fig. 2) foram utilizadas 2 placas GY-521, cujo microchip, MPU6050, integra um sensor de acelerômetro e de giroscópio com 6 eixos distintos, sendo *yaw*, *pitch* e *roll* do giroscópio e x, y e z do acelerômetro, além de dispor de um sensor de temperatura embutido. Entretanto, no sistema de controle abordado no presente estudo, foram explorados apenas os eixos *Yaw* e *Roll* do giroscópio do sensor acoplado à luva, que foram responsáveis por executarem os movimentos de rotação e de elevação/depressão do manipulador, respectivamente, e o eixo *Roll* do giroscópio do sensor localizado no antebraço, que foi responsável pelos movimentos de avanço/recuo do braço robótico. Além disso, foi usado um sensor resistivo flexível *Sparkfun* de 2,2" no dedo indicador da luva para controlar o efetuator (garra pegadora).

Com o intuito de fazer com que os movimentos do manipulador mecânico fossem mais estáveis e com maior precisão, fez-se o uso do *Digital Motion Processor* (DMP). O DMP é fornecido pela *InvenSense, Inc.*, a fornecedora da tecnologia do sensor inercial utilizado. O dispositivo MPU6050 combina o giroscópio de 3 eixos e o acelerômetro de 3 eixos em uma mesma matriz de silício, juntamente com um DMP integrado que processa complexos algoritmos. (Fig. 3) (INVENSENSE, 2020). Essa ferramenta computa os dados em quaterniões que são conjuntos numéricos que descrevem a rotação em 4 valores numéricos, sendo que 3 deles fornecem as coordenadas para o eixo da rotação, enquanto o quarto valor é determinado pelo ângulo rotacionado. A utilização dos quaterniões em detrimento dos ângulos de Euler destaca-se por eliminar o efeito do *gimbal lock*, que consiste na perda de um grau rotacional de

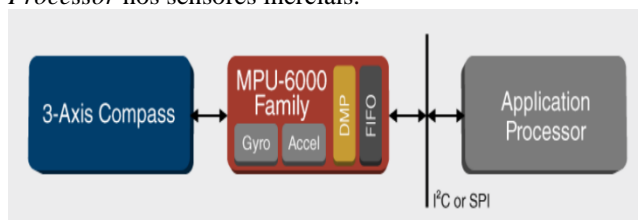
liberdade quando 2 dos 3 eixos se alinham (SHOEMAKE, 1985).

Figura 2: Disposição dos sensores da aparelhagem de controle na luva e no antebraço.



Fonte: Os autores, 2020.

Figura 3: Fluxograma operacional do *Digital Motion Processor* nos sensores inerciais.



Fonte: InvenSense, 2020.

A interferência de um ruído em sistemas eletrônicos pode ser definida de diversas maneiras. Dentre todas, uma de grande importância é a razão da potência do sinal de entrada e a potência do sinal de ruído ou apenas razão sinal/ruído (SNR). A SNR tem a finalidade de determinar a eficiência do processo, quanto maior for melhor será a qualidade do sinal (PADUA, 2014). Esse trabalho utilizou-se da Equação 1 como uma equação alternativa para calcular a SNR da variação dos valores do sensor MPU6050 com e sem a utilização do DMP.

$$SNR = \frac{\mu}{\sigma} \quad (1)$$

Onde:

SNR- relação sinal/ruído

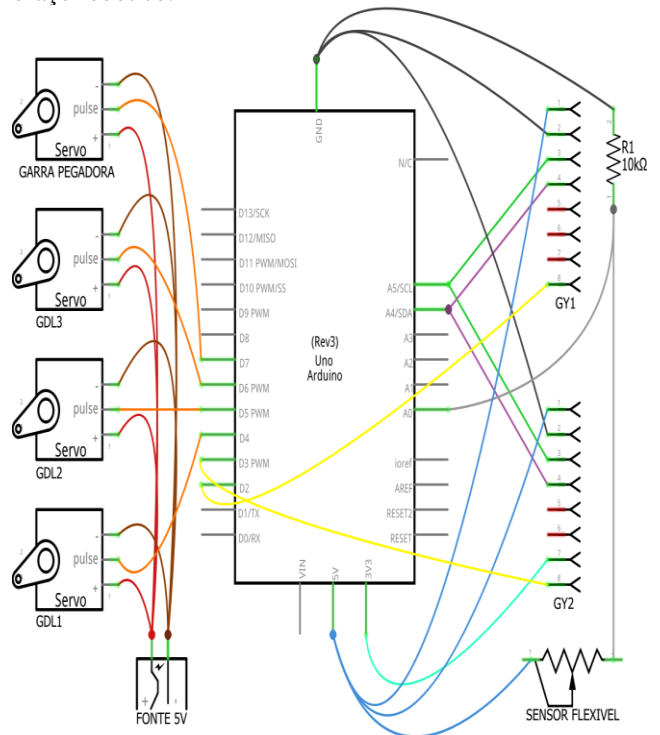
μ - média

σ - desvio padrão

Uma placa Arduino UNO R3 foi utilizada como micro controlador do sistema, controlando os movimentos dos atuadores (micros servomotores) de acordo com os

valores obtidos pelos sensores nos seus respectivos eixos (Fig. 4). Os códigos do controle do braço robótico com a utilização do DMP (Apêndice A) e sem a utilização do mesmo (Apêndice B) foram escritos em linguagem de programação C++ no software da Arduino IDE. Como processador, foi usado um *laptop* que possui o Windows 10 Pro de 64bits como sistema operacional.

Figura 4: Esquemático do circuito eletrônico de operação do braço robótico.



Fonte: Os autores, 2020.

RESULTADOS E DISCUSSÃO

O presente trabalho foi norteado pelo estudo sobre um método alternativo de manipulação que não fosse feito por execuções pré-programadas mas por movimentos ou gestos realizados por uma pessoa. Além disso, a implementação dessa interface de controle entre humano e braço robótico possui um interessante e conveniente aspecto intuitivo e adaptativo em operações em que tais fatores são primordiais.

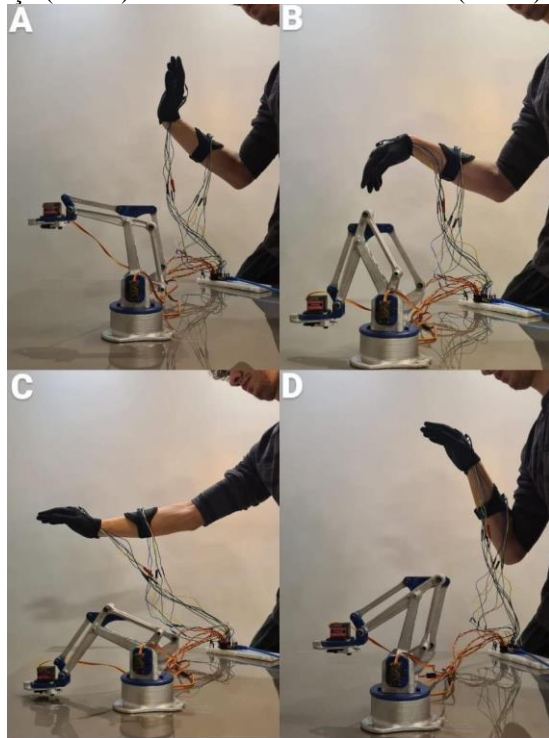
No contexto de controle realizado por sensores inerciais, os dados aferidos pelo acelerômetro e pelo giroscópio são frequentemente instáveis, fazendo com que a capacidade de prever os movimentos do braço robótico seja baixa. Para reduzir essa instabilidade, é necessário a introdução de filtros com complexas combinações algorítmicas à programação de extração de valores desses sensores.

O manipulador robótico foi testado conforme a execução de cada movimento do braço e da mão do operador. Os 3 eixos dos giroscópios dos sensores MPU6050, sendo os eixos *Yaw* e *Roll* do sensor da luva e o

eixo *Roll* do sensor posicionado no antebraço realizaram o controle coordenadamente com os graus de liberdade do manipulador. Na Fig. 5 foram representados os movimentos de elevação/depressão e de avanço/recuo do manipulador correspondentes ao eixo *Roll* do sensor da luva e do sensor no antebraço, respectivamente.

A fim de avaliar a eficiência e demonstrar os resultados da utilização do DMP na leitura de dados do giroscópio e do acelerômetro no MPU6050, foi realizada a coleta da variação dos valores do eixo *Roll* na execução do avanço e do recuo do braço robótico com e sem a integração do DMP no programa em um determinado período de tempo. A comparação da forma com que esses valores variam nos dois casos é demonstrada no gráfico apresentado na Fig. 6.

Figura 5: Representação dos movimentos de 2 dos 3 graus de liberdade existentes no sistema de controle do braço robótico. Em A: Movimento de elevação (GDL1); em B: Movimento de depressão (GDL1); em C: Movimento de avanço (GDL2) e em D: Movimento de recuo (GDL2).

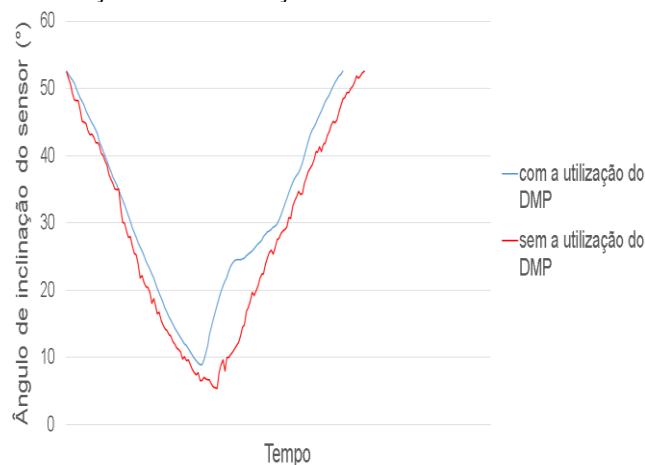


Fonte: Os autores, 2020.

A partir de uma análise visual do gráfico da Fig. 6, observa-se que os valores do eixo *Roll* do giroscópio do MPU6050 variam com uma constância e com uma linearidade muito maior com a utilização do DMP do que sem o uso do mesmo. Ainda, em posse dos dados do mesmo gráfico, podemos calcular o valor da SNR. O resultado da SNR sem a utilização do DMP foi de 1,90, valor inferior ao de quando utilizado o processador integrado, 2,40. Esse resultado evidencia que o DMP funciona como um filtro que minimiza os erros e ruídos inerentes ao sensor MPU6050,

garantindo, assim, movimentos precisos e suaves ao braço robótico.

Figura 6: Variação dos valores do MPU6050 no movimento de avanço e recuo do braço robótico.



Fonte: Os autores, 2020.

No programa em que foi incluso o DMP existem complexas operações matemáticas, como a combinação dos dados do acelerômetro e do giroscópio extraídos do sensor e a representação desses valores na forma de matrizes (quaterniões). Devido a isso, a placa controladora utilizada (Arduino UNO R3) não possui a velocidade de *Clock* ou a memória *flash* suficientes para executar a programação com DMP de forma perdurável. Portanto, o programa apresentou algumas instabilidades em que sua execução cessava aleatoriamente, tornando-se necessária uma reinicialização na programação sempre que essa paralisação ocorria.

CONCLUSÃO

Esse artigo foi capaz de desenvolver uma metodologia instintiva e coordenada de manipulação robótica com o uso de sensores inerciais, além de evidenciar as vantagens e o contexto da utilização de IMU's na implementação de teleoperações de braços robóticos. O método gestual de controle de manipuladores apresentado possui relevantes aplicações em atividades que exigem precisão e movimentos intuitivos, como operações cirúrgicas minimamente invasivas e desarme de explosivos.

Para trabalhos futuros, poderá ser feita a troca da placa controladora utilizada por uma que possui maior capacidade de processamento e de memória, como a Arduino Due para que o programa com DMP seja executado sem quaisquer interrupções ou instabilidades. Ainda, poderá ser incorporado ao sistema de controle comunicação via radiofrequência (RF) com a introdução de módulos receptores e transmissores RF e de uma placa controladora adicional com a finalidade de fazer com que o alcance de controle do braço robótico não seja limitado pela extensão dos fios que conectam as entradas analógicas.

REFERÊNCIAS

BAUER, Andreas. **Foreword**, 2019. Disponível em: <https://ifr.org/downloads/press2018/Foreword_WR_2019_Industrial_Robots.pdf>. Acesso em: 17 set. 2020.

CÉSAR, Danilo Rodrigues. Robótica livre: Robótica educacional com tecnologias livres. **Fórum Internacional de Software Livre**, v. 1, p. 1-6, 2005.

FANG, Bing; SUN, Fuchun; LIU, Huaping; GUO, Di. A novel data glove using inertial and magnetic sensors for motion capture and robotic arm-hand teleoperation. **Industrial Robot: An International Journal**, Beijing, Ch., v. 44, n. 2, p. 155-165, mar. 2017.

FIELD, Matthew; PAN, Zengxi; STIRLING, David; NAGHDY, Fazel. Human motion capture sensors and analysis in robotics. **Industrial Robot: An International Journal**, Wollongong, Au. v. 38, n. 2, p. 163-171, mar. 2011.

FILIFELOP. Micro Servo MG90S TowerPro. **Filipeflop**, 2020. Disponível em: <<https://www.filipeflop.com/produto/micro-servo-mg90s-towerpro/>>. Acesso em: 17 set. 2020.

FORHAN, Neisy Amparo Escobar. Giroscópios MEMS. **Instituto Nacional de Pesquisas Espaciais**, v. 80, p. 1.25, 2010.

INVENSENSE. MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Devices. **Invensense**, 2020. Disponível em: <<https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/>>. Acesso em: 30 out. 2020.

PADUA, Fabiano João Leoncio de. **Alocação de modem PLC utilizando SNR em uma rede elétrica de baixa tensão**. 2014. 79 f. Tese (Doutorado) – Pós-Graduação em Engenharia Elétrica, Universidade Estadual Paulista “Júlio de Mesquita Filho”, Ilha Solteira, 2014.

SHOEMAKE, Ken. Animating rotation with quaternion curves. **ACM SIGGRAPH Computer Graphics**, San Francisco, EUA, v. 19, n. 3, p. 245-254, jul. 1985.

SEKHAR, Rahul; MUSALAY, Rakesh Kiran; KRISHNAMURTHY, Yashwanth; B, Shreenivas. Inertial sensor based wireless control of a robotic arm. **IEEE International Conference on Emerging Signal Processing Applications**, Bangalore, In., p. 1-4, feb. 2012.

TORQUATO, Júlio César. Manipulação de braços a partir da detecção de mão humana por meio de visão de máquina. In: Mostra Nacional de Robótica, 2017, João Pessoa. **Anais...** Paraíba: IFPB, p. 1-6, 2017.

VARUM, Daniel António Teixeira. **Sistema de realidade virtual (Oculus Rift) para controlo remoto de braço robótico**. 2015. 41 f. Dissertação (Mestrado) – Programa de Pós-Graduação Integrado em Engenharia Informática e Computação, Faculdade de Engenharia da Universidade do Porto, Porto, 2015.

APÊNDICE

Apêndice A – Código- fonte da programação de controle do braço robótico com a utilização do *Digital Motion Processor* (DMP).

```
#include <Servo.h>

#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
// in "MPU6050_6Axis_MotionApps20.h"
// I'm now using the default value on line 305 to: 0x02, 0x16, 0x02, 0x00, 0x01 // D_0_22 inv_set_fifo_rate
// Correcting the PID code fixed my issues with the fifo buffer being too fast
#include "Wire.h"

MPU6050 mpu1(0x68);
MPU6050 mpu2(0x69);
// These are my MPU6050 Offset numbers: for mpu.setXGyroOffset()
// supply your own gyro offsets here, scaled for min sensitivity use MPU6050_calibration.ino <<< download to calibrate
// your MPU6050 put the values the program returns below
//      XA  YA  ZA  XG  YG  ZG
int MPUOffsets1[6] = { -858, -710, 1464, 29, 104, 44};
int MPUOffsets2[6] = { 274, -724, 1344, 14, 96, 7};

#define SERVO1 4
#define SERVO2 5
#define SERVO3 6
#define SERVO4 7

Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;

int DOF1;
int DOF2;
int DOF3;

int gripper = A0;
int gripperMov;

#define LED_PIN 13 //

// =====
// ===      INTERRUPT DETECTION ROUTINE      ===
// =====
volatile bool mpuInterrupt1 = false; // indicates whether MPU interrupt pin has gone high
void dmpDataReady1() {
    mpuInterrupt1 = true;
}

volatile bool mpuInterrupt2 = false; // indicates whether MPU interrupt pin has gone high
void dmpDataReady2() {
    mpuInterrupt2 = true;
}

// =====
// ===      MPU DMP SETUP      ===
// =====
```

```

int FifoAlive1 = 0; // tests if the interrupt is triggering
int FifoAlive2 = 0;
int IsAlive1 = -20; // counts interrupt start at -20 to get 20+ good values before assuming connected
int IsAlive2 = -20;
// MPU control/status vars
uint8_t mpu1IntStatus; // holds actual interrupt status byte from MPU
uint8_t mpu2IntStatus;
uint8_t dev1Status; // return status after each device operation (0 = success, !0 = error)
uint8_t dev2Status;
uint16_t packetSize1; // expected DMP packet size (default is 42 bytes)
uint16_t packetSize2;
uint16_t fifoCount1; // count of all bytes currently in FIFO
uint16_t fifoCount2;
uint8_t fifoBuffer1[64]; // FIFO storage buffer
uint8_t fifoBuffer2[64];

// orientation/motion vars
Quaternion q1; // [w, x, y, z] quaternion container
Quaternion q2;
VectorInt16 aa1; // [x, y, z] accel sensor measurements
VectorInt16 aa2;
VectorInt16 aaReal1; // [x, y, z] gravity-free accel sensor measurements
VectorInt16 aaReal2;
VectorInt16 aaWorld1; // [x, y, z] world-frame accel sensor measurements
VectorInt16 aaWorld2;
VectorFloat gravity1; // [x, y, z] gravity vector
VectorFloat gravity2;
float euler1[3]; // [psi, theta, phi] Euler angle container
float euler2[3];
float ypr1[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector
float ypr2[3];
float Yaw1, Pitch1, Roll1; // in degrees
float Yaw2, Pitch2, Roll2;

void MPU6050Connect() {
    static int MPUInitCntr1 = 0;
    static int MPUInitCntr2 = 0;
    // initialize device
    mpu1.initialize(); // same
    mpu2.initialize();
    // load and configure the DMP
    dev1Status = mpu1.dmpInitialize();// same
    dev2Status = mpu2.dmpInitialize();

    if (dev1Status != 0) {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)

        char * StatStr1[5] { "No Error", "initial memory load failed", "DMP configuration updates failed", "3", "4"};

        MPUInitCntr1++;

        //Serial.print(F("MPU connection Try #"));
        //Serial.println(MPUInitCntr1);
        //Serial.print(F("DMP Initialization failed (code "));

```

```
//Serial.print(StatStr1[dev1Status]);
//Serial.println(F(""));

if (MPUInitCnt1 >= 10) return; //only try 10 times
delay(1000);
MPU6050Connect(); // Lets try again
return;
}
if (dev2Status != 0) {
// ERROR!
// 1 = initial memory load failed
// 2 = DMP configuration updates failed
// (if it's going to break, usually the code will be 1)

char * StatStr2[5] { "No Error", "initial memory load failed", "DMP configuration updates failed", "3", "4"};

MPUInitCnt2++;

//Serial.print(F("MPU connection Try #"));
//Serial.println(MPUInitCnt2);
//Serial.print(F("DMP Initialization failed (code ")");
//Serial.print(StatStr2[dev2Status]);
//Serial.println(F(""));

if (MPUInitCnt2 >= 10) return; //only try 10 times
delay(1000);
MPU6050Connect(); // Lets try again
return;
}

mpu1.setXAccelOffset(MPUOffsets1[0]);
mpu1.setYAccelOffset(MPUOffsets1[1]);
mpu1.setZAccelOffset(MPUOffsets1[2]);
mpu1.setXGyroOffset(MPUOffsets1[3]);
mpu1.setYGyroOffset(MPUOffsets1[4]);
mpu1.setZGyroOffset(MPUOffsets1[5]);

mpu2.setXAccelOffset(MPUOffsets2[0]);
mpu2.setYAccelOffset(MPUOffsets2[1]);
mpu2.setZAccelOffset(MPUOffsets2[2]);
mpu2.setXGyroOffset(MPUOffsets2[3]);
mpu2.setYGyroOffset(MPUOffsets2[4]);
mpu2.setZGyroOffset(MPUOffsets2[5]);

//Serial.println(F("Enabling DMP..."));
mpu1.setDMPEnabled(true);
mpu2.setDMPEnabled(true);
// enable Arduino interrupt detection

//Serial.println(F("Enabling interrupt detection (Arduino external interrupt pin 2 and pin 3 on the Uno)..."));
attachInterrupt(digitalPinToInterrupt(2), dmpDataReady1, FALLING); //pin 2 on the Uno
attachInterrupt(digitalPinToInterrupt(3), dmpDataReady2, FALLING); //pin 3 on the Uno

mpu1IntStatus = mpu1.getIntStatus(); // Same
mpu2IntStatus = mpu2.getIntStatus();
// get expected DMP packet size for later comparison
packetSize1 = mpu1.dmpGetFIFOPacketSize();
packetSize2 = mpu2.dmpGetFIFOPacketSize();
```



```

delay(1000); // Let it Stabalize
mpu1.resetFIFO(); // Clear fifo buffer
mpu2.resetFIFO();
mpu1.getIntStatus();
mpu2.getIntStatus();
mpuInterrupt1 = false;
mpuInterrupt2 = false;

}
// =====
// ===          i2c SETUP Items          ===
// =====

void i2cSetup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif
}

void setup() {
    Serial.begin(9600); //115200
    while (!Serial);

    i2cSetup();

    //Serial.println(F("Alive"));
    MPU6050Connect();
    pinMode(LED_PIN, OUTPUT); // LED Blinks when you are recieving FIFO packets from your MPU6050

    servo1.attach(SERVO1);
    servo2.attach(SERVO2);
    servo3.attach(SERVO3);
    servo4.attach(SERVO4);

    servo1.write(0);
    servo2.write(0);
    servo3.write(0);
    servo4.write(0);
}

int sensorFlex = 0;
int sensorFlexDegrees = 0;
static long QTimer = 0;

uint16_t MaxPackets1 = 0;
uint16_t MaxPackets2 = 0;

void loop() {
    if (mpuInterrupt1 && mpuInterrupt2) { // wait for MPU interrupt or extra packet(s) available
        GetDMP(); // Gets the MPU Data and canculates angles
    }

    //LEFT/RIGHT Movement
    DOF1 = map(Yaw1,0,90,90,0);

```

```

servo1.write(DOF1);

//UP/DOWN Movement
DOF2 = map(Roll1,0,60,0,60);
servo2.write(DOF2);

//FORWARD/BACKWARD Movement
DOF3 = map(Roll2,45,0,0,60);
servo3.write(DOF3);

//GRIPPER
sensorFlex = analogRead(gripper);
sensorFlexDegrees = map(sensorFlex, 150, 200, 0, 90);
if(sensorFlexDegrees > 50)
{
    gripperMov = 0;
}
else
{
    gripperMov = 90;
}
servo4.write(gripperMov);

QTimer = millis();
if ((long)( millis() - QTimer ) >= 100) {
    QTimer = millis();
    Serial.print(F("\t Yaw1 ")); Serial.print(Yaw1);
    Serial.print(F("\t Pitch1 ")); Serial.print(Pitch1);
    Serial.print(F("\t Roll1 ")); Serial.print(Roll1);
    Serial.print(F("\t Yaw2 ")); Serial.print(Yaw2);
    Serial.print(F("\t Pitch2 ")); Serial.print(Pitch2);
    Serial.print(F("\t Roll2 "));
    Serial.print(Roll2);
    Serial.println();
}
}

void GetDMP() { // Best version I have made so far
// Serial.println(F("FIFO interrupt at:"));
// Serial.println(micros());
mpuInterrupt1 = false;
mpuInterrupt2 = false;
FifoAlive1 = 1;
FifoAlive2 = 1;
fifoCount1 = mpu1.getFIFOCount();
fifoCount2 = mpu2.getFIFOCount();
/*
fifoCount is a 16-bit unsigned value. Indicates the number of bytes stored in the FIFO buffer.
This number is in turn the number of bytes that can be read from the FIFO buffer and it is
directly proportional to the number of samples available given the set of sensor data bound
to be stored in the FIFO
*/

// PacketSize = 42; reference in MPU6050_6Axis_MotionApps20.h Line 527
// FIFO Buffer Size = 1024;
MaxPackets1 = 20; // 20*42=840 leaving us with 2 Packets (out of a total of 24 packets) left before we overflow.

```



```
mpu2.dmpGetYawPitchRoll(ypr2, &q2, &gravity2);  
Yaw2 = (ypr2[0] * 180 / M_PI);  
Pitch2 = (ypr2[1] * 180 / M_PI);  
Roll2 = (ypr2[2] * 180 / M_PI);  
}
```


APÊNDICE B – Código- fonte da programação de controle do braço robótico sem a utilização do *Digital Motion Processor* (DMP).

```
#include <Servo.h>
#include <Wire.h>

#define SERVO1 4
#define SERVO2 5
#define SERVO3 6
#define SERVO4 7

const int MPU_addr1=0x68,MPU_addr2=0x69;
int16_t axis_X1,axis_Y1,axis_Z1;
int16_t axis_X2,axis_Y2,axis_Z2;

int minVal=265;
int maxVal=402;

double x1;
double y1;
double z1;

double x2;
double y2;
double z2;

Servo servo1;
Servo servo2;
Servo servo3;
Servo servo4;

int DOF1R;
int DOF1L;
int DOF2;
int DOF3;

int gripper = A0;
int gripperMov;

int size = *( &axis_X1 + 1 ) - axis_X1;
void setup(){

  Wire.begin();                      //begin the wire communication
  Wire.beginTransmission(MPU_addr1); //begin, send the slave adress (in this case 68)
  Wire.write(0x6B);                  //make the reset (place a 0 into the 6B register)
  Wire.write(0);
  Wire.endTransmission(true);        //end the transmission

  Wire.begin();
  Wire.beginTransmission(MPU_addr2);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);

  Serial.begin(9600);

  servo1.attach(SERVO1);
  servo2.attach(SERVO2);
```

```

servo3.attach(SERVO3);
servo4.attach(SERVO4);

servo1.write(0);
servo2.write(0);
servo3.write(0);
servo4.write(0);
}

void GetAngle1(const int MPU) {

Wire.beginTransmission(MPU_addr1);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr1,14,true);

axis_X1=Wire.read()<<8|Wire.read();
axis_Y1=Wire.read()<<8|Wire.read();
axis_Z1=Wire.read()<<8|Wire.read();

int xAng = map(axis_X1,minVal,maxVal,-90,90);
int yAng = map(axis_Y1,minVal,maxVal,-90,90);
int zAng = map(axis_Z1,minVal,maxVal,-90,90);
x1= RAD_TO_DEG * (atan2(-yAng, -zAng)+PI);
y1= RAD_TO_DEG * (atan2(-xAng, -zAng)+PI);
z1= RAD_TO_DEG * (atan2(-yAng, -xAng)+PI);

}

void GetAngle2(const int MPU) {

Wire.beginTransmission(MPU_addr2);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU_addr2,14,true);

axis_X2=Wire.read()<<8|Wire.read();
axis_Y2=Wire.read()<<8|Wire.read();
axis_Z2=Wire.read()<<8|Wire.read();

int xAng = map(axis_X2,minVal,maxVal,-90,90);
int yAng = map(axis_Y2,minVal,maxVal,-90,90);
int zAng = map(axis_Z2,minVal,maxVal,-90,90);
x2= RAD_TO_DEG * (atan2(-yAng, -zAng)+PI);
y2= RAD_TO_DEG * (atan2(-xAng, -zAng)+PI);
z2= RAD_TO_DEG * (atan2(-yAng, -xAng)+PI);

Serial.print("g1:\t");
Serial.print(x1); Serial.print("\t");
Serial.print(y1); Serial.print("\t");
Serial.print(z1); Serial.print("\t");
Serial.print("g2:\t");
Serial.print(x2); Serial.print("\t");
Serial.print(y2); Serial.print("\t");
Serial.print(z2); Serial.print("\t");
Serial.println();

}

```

```
void loop(){

  GetAngle1(MPU_addr1);
  GetAngle2(MPU_addr2);

  //LEFT Movement
  if(y1 >= 0 && y1 <= 60){
    DOF1L = map(y1, 60, 0, 180, 90);
    servo1.write(DOF1L);
  }

  //RIGHT Movement
  if(y1 <= 356 && y1 >= 290){
    DOF1R = map(y1, 356, 290, 90, 0);
    servo1.write(DOF1R);
  }

  //UP/DOWN Movement
  if(x1 <= 60 && x1 >= 0){
    DOF2 = map(x1, 0, 60, 0, 110);
    servo2.write(DOF2);
  }

  //FORWARD/BACKWARD Movement
  if(x2 <= 45 && x2 >= 0){
    DOF3 = map(x2, 45, 0, 0, 90);
    servo3.write(DOF3);
  }

  //GRIPPER
  int sensorFlex = analogRead(gripper);
  int sensorFlexDegrees = map(sensorFlex, 150, 200, 0, 90);
  if(sensorFlexDegrees > 50)
  {
    gripperMov = 0;
  }
  else
  {
    gripperMov = 90;
  }
  servo4.write(gripperMov);

  delay(100);
}
```