Dokumentation für

Berechnung von Pi

der **Juventus Technikerschule HF**

**Inhaltsverzeichnis:**

[1 GMPY Rechenmethode 3](#_Toc117511613)

[2 Taskbeschreibung 3](#_Toc117511614)

[2.1 Struktogramm 3](#_Toc117511615)

[2.2 EventBits und TaskNotifications 3](#_Toc117511616)

[3 Zeitmessungen 3](#_Toc117511617)

[3.1 Resultate 3](#_Toc117511618)

[3.2 Geschwindigkeitsvergleich 3](#_Toc117511619)

[4 Rechenleistung 3](#_Toc117511620)

# Asin Rechenmethode

Der Asin- Algorithmus oder Arcussinusfunktion ist die umgekehrte Funktion der Sinusfunktion also und wird somit aus der Kreisbogenfunktion hergeleitet.

Bei wenn der Variable x in der funktion der Wert 1 zugewiesen wird, ergibt sich dadurch .

In der Software wird dies mithilfe der math.h library realisiert.

Die Umsetzung der berechnung sieht wie folgt aus: . pi\_calc stellt die pi variable dar, die danach als double am Bildschirm ausgegeben wird.

Ein starker Nachteil dieses Algorithmus ist die niedrige Auflösung, die somit ein weniger genaues resultat liefert.

Dieses Resultat nähert sich über längere Rechenzeit ebenfalls nicht wie der Leibniz Algorithmus Pi, sondern bleibt immer auf dem gleichen fix erechneten Wert.

Vorteile dieses Algorithmus ist die reduzierte Rechenzeit und Leistung, die für ein Resultat benötigt wird wenn man diese mit dem Leibnizalgorithmus vergleicht.

Chart, line chart

Description automatically generated

# Code und Task Abläufe

Die Software wurde auf die Weise geschrieben, die für alle die mit der Syntax vertraut sind verständlich ist.

Der Code ist in 5 Blöcke aufgeteilt von denen 4 als Tasks definiert wurden. Eine «state-machine» Funktion wurde dafür verwendet zwischen den individuellen Tasks umschalten zu können und an Rechenleistung zu sparen.

Um die Abläufe innerhalb der Software etwas strukturierter darstellen, meine Vorgehensweise festhalten und den Aufbau der Software auch dem Laien näherbringen zu können habe ich dazu ein Struktogramm erstellt. Dieses Struktogramm soll es einem Programmierer oder User ermöglichen möglichst schnell das Konzept des Programms zu verstehen, auch wenn dieser sich nicht mit der Syntax vertraut ist.

Die Software basiert Zeile um Zeile auf den Aufbau des Struktogramms.

## Text Description automatically generatedStruktogramme

Diagram

Description automatically generatedDer «main» block des Programms von dem alle Tasks und dessen Rechnpriotität aus ermittelt werden.

Der «Controller Task» Block ist für die Event-Bit zuteilung zuständig und wird durch die button inputs des Eduboards angesteuert.

Diagram

Description automatically generatedTable

Description automatically generatedGraphical user interface, application

Description automatically generatedDer „User Interface“ Block übernimmt alle interaktionen zwischen User und Software und teilt den Inputs die entsprechenden zustände zu. Ebenso ist dieser Block für einen Teil der Zeitmessung der zwei Algorithmen zuständig.

Die Leibniz und Asin Tasks enthalten beide ihre jeweiligen Algorithmen. Diese Tasks ermitteln Pi via ihrer zugeordneten Rechenmethode und messen auch die Zeit, die dessen Ausführung benötigt.

Der „asin task“ enthält ebenfalls einen „500ms Interrupt“ da der task nur einen durchlauf benötigt um Pi zu berechnen und somit nicht sehr viel Rechenleistung konsumiert.

# EventBits

Die EventBits werden dazu verwendet die Eingaben des Users festzuhalten.

Der Nachteil die reinen Button Inputs zu prüfen besteht in der Zeit die zwischen Auslösung der Button und Prüfen des Inputs versteichen kann. Da der UI-Task sowie die Algorithmus-tasks die auf die Inputs reagieren müssen nicht immer genau dann an der Reihe sind wenn der Knopf betätigt ist kann es sein, dass der Task den Input verpasst und somit nicht reagiert.

Mithilfe von EventBits kann ein Bit durch Betätigung der Buttons festgehalten werden und unabhängig weiterer Inputs durch die zuständige Button abfrage wieder gelöscht werden. Dies führt dazu, dass die inputs alle gespeichert werden und erst dann gelöscht werden wenn sie von der Software bearbeitet wurden.

Geschrieben werden die EventBits mit:

xEventGroupSetBits(egButtonEvents, BUTTON1\_SHORT);

Um den erwünschten Programmteil nicht mehrmals auszuführen müssen die EventBits auch wieder gelöscht werden. Dies geschieht folgendermassen:

xEventGroupClearBits(egButtonEvents, BUTTON1\_SHORT);

# Zeitmessungen

In C++ gibt es mehrere Methoden die Zeit die während eines Task Ablaufes verstreicht zu messen. Viele dieser methoden können sich jedoch durch die Baudrate, Taskinhalt und Ausführung im Programm beinflussen und verfälschen. Darum verwende ich die getTickcount-Funktion.

Die Freertos lib stellt diese Funktion bereit und kann sehr einfach in einem Task implementiert werden. Sie kann zwar auch die Zeit nicht genaustens messen aber man kann näherungsweise Berechnungszeiten eruieren, um jeweilige Rückschlüsse auf notwendige Rechenleistung schliessen zu können.

In der Software erhält zu beginn des Tasks Startzeitvariable ihren Wert:

leibnizstart = xTaskGetTickCount();

Um dann beim erreichen des erwünschten Werts die Zeit dem Bildschirm ausgeben zu können, wird die Endzeit gemessen und die Startzeit der Endzeit abgezogen. Umgesetzt wird dies folgendermassen.

if(pi\_calc < 3.1416)

{

*sprintf*(&countstring[0], "Total Time: %lums", leibniztime);

}

else

{

leibniztime = xTaskGetTickCount() - leibnizstart;

*sprintf*(&countstring[0], "Timer: %lu", leibniztime);

}

## Resultate

In der Aufgabenstellung wurde die gewünschte Genauigkeit für Pi bei 5 Stellen nach dem Komma definiert. Der Leibnizalgorithmus benötigte total 8-9 sekunden um diesen Wert zu erreichen, von denen pro durchlauf jeweils 160ms für etwaige «if» abfragen und start/stop prozesse verloren gingen.

Der Asin Algorithmus schnitt hierbei einiges besser ab und konnte nach nur 1 milisekunde die Berechnung abschliessen.

Ob die Zeit wirklich eine 1ms und nicht weniger entspricht ist sehr unwarscheinlich. Die getTickCount-Funktion kann jedoch scheinbar keine kleinere Masseinheit festhalten also bleibt diese Zeitmessung etwas spekulativ.

## Geschwindigkeitsvergleich

Das der Asin-Task seinen gewünschten Wert deutlich schneller als der Leibniz-Task erreicht ist ein erwartetes Resultat.

Mit absoluter Sicherheit kann der Code weiter optimiert werden um einen schnelleren Pi-Berechnungsprozess zu erreichen, aber der Unterschied zwischen dem Leibniz- und Asinalgorithmus wird nur schon durch den Rest der Software beeinflusst.

Der Asin-Task ist nach nur einem durchlauf abgeschlossen und hat keine weiteren schritte die ausgeführt werden müssen.

Der Leibnizalgorithmus muss hingegen den Leibniz-Task, UI-Task und Controller-Task mehrfach durchlaufen um am erwünschten Pi-Wert anzukommen. Gekoppelt mit viel häufigeren Abfragen und Bildschimaktualiesierungen wird die Berechnungszeit vor allem durch die niedrige Priorität des Leibniz-Tasks um ein Vielfaches in die Länge gezogen.

# Rechenleistung

Zur benötigte Rechenleistung lässt sich nicht viel aus den berechneten Rechenzeiten entnehmen. Rein theoretisch benötigen beide Algorithmen gleich viel reine Rechnenleistung, da im Falle eines Taskinterrupts wie beim Asin-Task einfach in der Zwischenzeit andere Tasks häufiger von der Software bearbeitet werden. Auch wenn keine effektive Aktivität stattfindet werden alle «ifs» und «state machine» immernoch abgefragt. Um in dem Bereich einen Unterschied feststellen zu können müsset man die Baudrate ändern.

Bei der Prozessorleistung ist aber ganz klar der Leibnizalgorithmus der Anspruchsvollste. Zur berechnung sind mehr Variablen und somit mehr Speicherplatz notwendig. Die Informationen in diesem Speicher werden auch durch das wiederholte Durchlaufen des Leibniz-Tasks im vergleich zu Asin-Task öfter umgeschrieben. Der Bedarf an Leistung wird hauptsächlich durch den Speicher beeinflusst.

Dieses Wissen wurde über Erfahrung angeeignet und wurde in keiner weise elektronisch gemessen. Eine Temparaturmessung des Prozessors fand über einen Fingertest am Prozessorgehäuse statt.

Dass das Gehäuse des Prozessors bei Berechnung des Leibnizalgorithmus wärmer wird, ist der Einzige beweis den ich für diese These erbringen kann.